MOTOROLA
intelligence everywhere™

digital dna™

Industrial CAN I/O Module

Designer Reference Manual

HCS12 Microcontrollers

DRM033/D
Rev. 0, 03/2003

MOTOROLA.COM/SEMICONDUCTORS

Industrial CAN I/O Module Reference Design

Designer Reference Manual — Rev 0

by: Jaromir Chocholac

Zdenek Kaspar

TU682

Czech Systems Laboratories

Revision history

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://www.motorola.com/semiconductors

The following revision history table summarizes changes contained in this document. For your convenience, the page number designators have been linked to the appropriate location.

Revision history

Date	Revision Level	Description	Page Number(s)
January, 2003	1.0	Initial release	N/A

List of Sections

Table of Contents

Section 1. Introduction

Section 2. Quick Start

Section 3. Hardware Description

Section 4. Software Module Descriptions

Appendix A. Source Code Files

Appendix B. Bill of Materials and Schematics

Appendix C. Glossary

List of Figures

List of Tables

Section 1. Introduction

1.1 Contents

1.2 Application intended functionality

This reference design of the Industrial CAN I/O Module provides an Input/ Output module for industrial automation purpose. The reference design demonstrates the flexibility of Motorola analog products portfolio and the ability of HCS12 MCU family to control analog devices trough the CAN bus. The CAN interface is compatible to ISO 11898 and allow maximum data transfer rate of 0,5 Mbit/s.

The reference design is based on available analog devices portfolio to realize digital inputs (MC33884), outputs (MC33298) and CAN transceiver (PC33394 or MC33389). The application is supported by HCS12 MCU.

According to the functionality of the module, the firmware includes functions for switching the outputs and for polling the inputs. The status information of the I/O's is transferred at programmable cycle times onto the CAN bus.

1.3 Benefits of our solution

The Industrial CAN I/O module uses modular concept. The base board with the microcontroller, analog inputs and CAN interface is able to drive optional I/O boards through the SPI interface. Besides this, the Industrial CAN I/O Module can be used as a hardware platform for high level communication protocol software development. In addition that, the Industrial CAN I/O Module enables the implementation and testing of user software.

The module is designed to be housed in a compact DIN enclosure.

Section 2. Quick Start

2.1 Contents

2.2 Introduction

This section describes the main procedures required to set up and start the Industrial CAN I/O Module Demo System. The demo is designed to show the basic functionality of the Industrial CAN I/O Module. The document also describes the specific steps and provide additional reference information.

The PC Master software is used to control and observe application variables. This tool allows the remote control of an application from a user-friendly graphical environment, running on a PC. It also enables the real-time display of application variables, in both textual and graphical form.

The Industrial CAN I/O Module kit is distributed with the following components:

- Industrial CANI/O Module
- PC_CAN Interface

- Black Box
- CD ROM with PC Master software
- CAN bus Cable

You can see the Demo System layout in **Figure 2-1**.

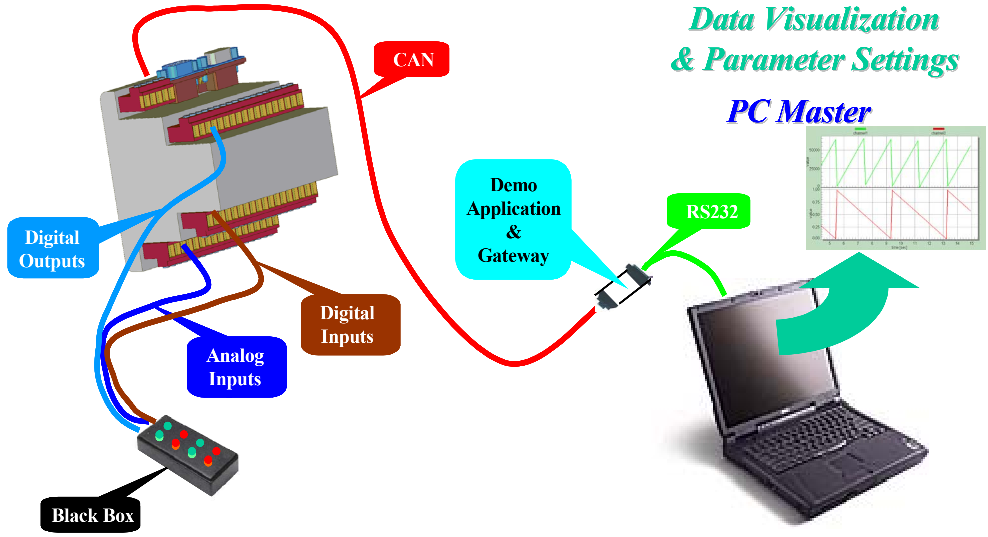


Figure 2-1. Demo System Layout

2.3 System Requirements

The Industrial CAN I/O Module and PC_CAN Interface are distributed with embedded demo application software. No additional software is needed for the demonstration purposes.

The PC Master software application can run on any computer using Microsoft Windows NT, 98 or 95 (+DCOM) operating systems, with installed Internet Explorer V 4.0 or higher installed.

Overall requirements, including those for the Internet Explorer 4.0 or higher, are as follows:

Computer: 486DX/66 MHz or higher processor (Intel Pentium recommended)

Operating system: Microsoft Windows NT, Windows 98 or Windows 95 + DCOM pack

Memory: Windows 95 or 98: 16 Mb RAM minimum (32 Mb recommended); for Windows NT: 32 Mb RAM minimum (64 Mb recommended)

Required software: Internet Explorer 4.0 or higher installed.

For selected features (e.g. regular expression-based parsing), Internet Explorer 5.5 or higher is required.

Hard drive space: 8 MB

Other hardware requirements: Mouse, serial RS-232 port for local control, network access for remote control

2.4 PC Master Software Installation

The PC Master software is distributed on a CD ROM.

To install the PC Master software, follow these steps:

1. Insert the CD into your CD-ROM drive.
2. Click on the CD-ROM drive; click on the PC Master folder.
3. Double click on **pcm12-11.exe** file.
4. Follow the on-screen instructions and answer the prompted questions.
5. Copy **CANIODEMO** folder to the PC hard drive.

2.5 Black Box

The aim of the Black Box is to simulate a controlled device. Two controlled channels are built into the Black Box. Each channel consists of two buttons connected to digital inputs, two LEDs connected to digital outputs, and an electrolytic capacitor connected to an analog input and a digital output. The capacitor is charged through a resistor from a 24V supply and discharged through a digital output. The process is controlled through communication between the Industrial CAN I/O Module and an application running on the PC_CAN Interface. The demo performance is very simple. The capacitor is charged and the voltage is monitored by the analog channel. When the voltage reaches the defined value, the capacitor is discharged through the digital output. The process through each separate channel can be started and stopped by pressing the relevant *START* and *STOP* buttons. A picture of the Black Box is shown in **Figure 2-2**.

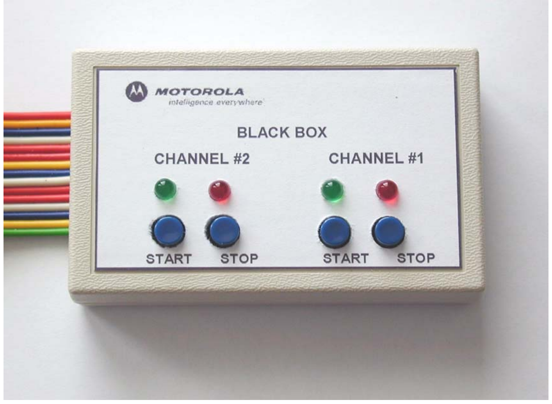


Figure 2-2. Black Box

2.6 Demo System Setup

The Industrial CAN I/O Module must have a Node ID. The Module provides an 8-position DIP switch (SW1) on the Base Board for configuring the Node ID, as well as the CAN speed. See **Figure 2-3**. Each Module is distributed with a Node ID that relates to the Module serial number and the CAN speed set to 500 kbps as default. Typically, the DIP switch positions should be left in this default configuration. If alternate system settings are required, refer to sections **2.6.1** and **2.6.2**.

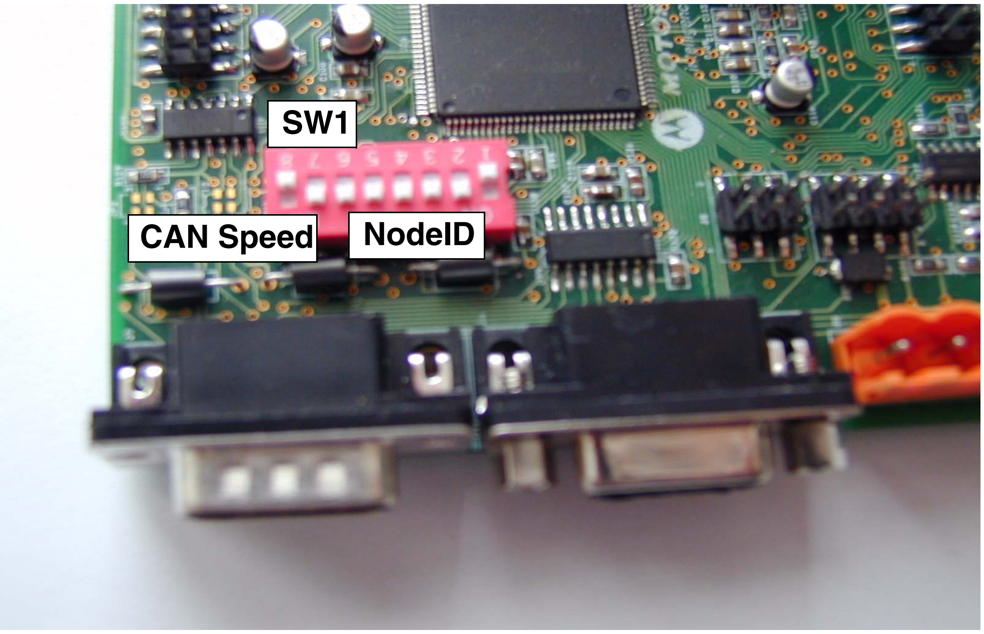


Figure 2-3. DIP Switch SW1

The following cabling must be completed before the Industrial CAN I/O Module Demo System is started:

1. Connect a straight-through serial cable from the PC to the PC_CAN Interface.
2. Connect a straight-through CAN cable from the PC_CAN Interface to the Module's CAN connector.
3. Connect the Black Box to the Module. See
4. Connect the 24V Power Supply. All of the indication LEDs on the Module and the Black Box must flash. The completed Demo connection is shown in **Figure 2-5**.

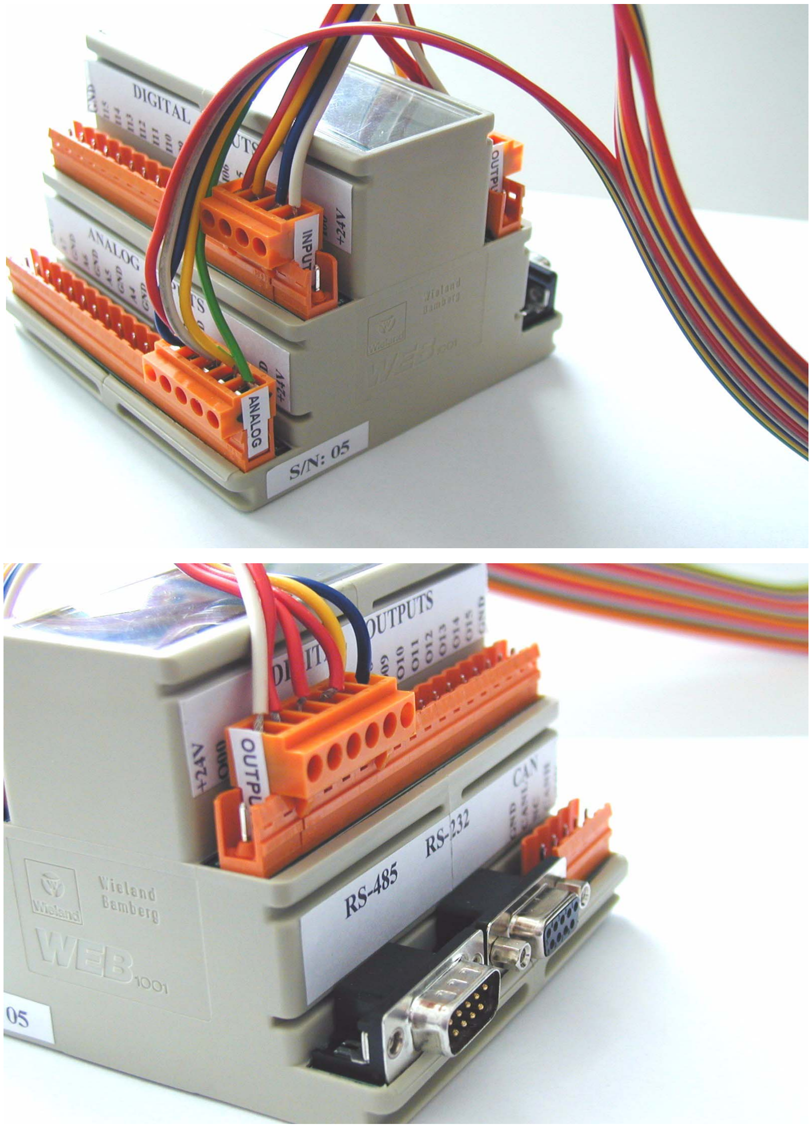
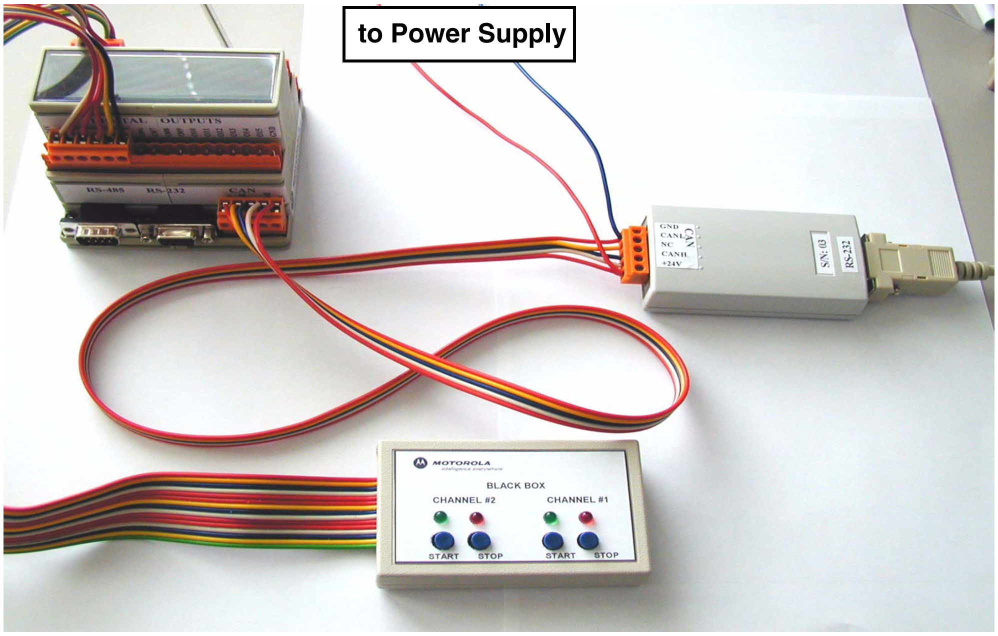


Figure 2-4. Black Box Cable to Module Connection



to Power Supply

Figure 2-5. Demo Connection

2.6.1 Setting the Node ID

The Node ID is set using positions 1 to 6 of the DIP switch SW1 on the Base Board. **Table 2-1** gives weight meaning of the SW1 positions.

Table 2-1. SW1 positions weight

SW1 position	1	2	3	4	5	6
Weight (if OFF)	1	2	4	8	16	32

As an example, the SW1 settings for the Node ID value 43, are shown in **Table 2-2**.

Table 2-2. SW1 settings

SW1 position	1	2	3	4	5	6
Position status	OFF	OFF	ON	OFF	ON	OFF
Weight	1	2	0	8	0	32

2.6.2 Setting the CAN bit rate

The Industrial CAN I/O Module supports three different bit rates (125, 250 and 500kbps), selected by setting positions 7 and 8 of the DIP switch SW1.

Table 2-3 shows the DIP switch settings for the supported rates.

Table 2-3. DIP Switch settings

SW1 position	7	8	bit rate
Position status	ON	ON	125kbps
	OFF	ON	250kbps
	ON	OFF	500kbps
	OFF	OFF	Not supported default value 125kbps

NOTE: The application demo software only supports a CAN bit rate 500kbps.

2.7 Starting the Demo System

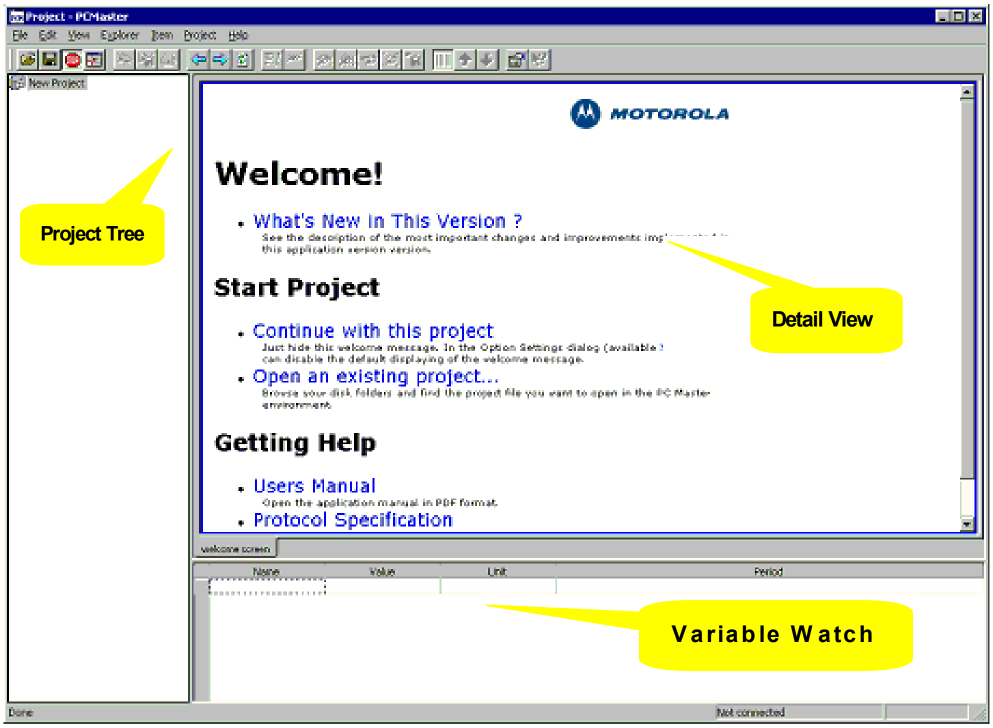
This section describes step by step how to start the Demo System. Once you have installed the PC Master software, finished the cabling and connected the power supply, you can start the PC Master program.

2.7.1 Step 1

Start the PC Master program by selecting the menu sequence

Start -> Programs -> PC Master 1.2.

When the program starts, the screen displays the main application window. If there is no project currently loaded, the window displays a welcome page, as shown in Figure 2-6.



Welcome!

- What's New in This Version ?
  See the description of the most important changes and improvements imp
  this application version version.

Start Project

- Continue with this project
  Just hide this welcome message. In the Option Settings dialog (available
  can disable the default displaying of the welcome message.
- Open an existing project...
  Browse your disk folders and find the project file you want to open in the PC Master
  environment.

Getting Help

- Users Manual
  Open the application manual in PDF format.
- Protocol Specification

Figure 2-6. Main Window of PC Master Software

The application window consists of three panes:

the *Project Tree*, the *Detail View* and the *Variable Watch*.

The *Project Tree* pane.

Contains a logical tree structure of the application being monitored/controlled. Users can add project sub-blocks, Scope, and Recorder definitions to the project block, in a logical structure, to form a *Project Tree*. This pane provides point and click selection of defined *Project Tree* elements.

The *Detail View* pane.

Dynamically changes its contents depending on the item selected in the *Project Tree*. You can find detailed information about this pane in the *PC Master Software User Manual.*

The *Variable Watch* pane.

Contains the list of variables assigned to the watch. It displays the current variable values and allows you to change them (if enabled in the variable definition).

All the information related to one application is stored in a single project file with the extension ".pmp".

2.7.2 Step 2

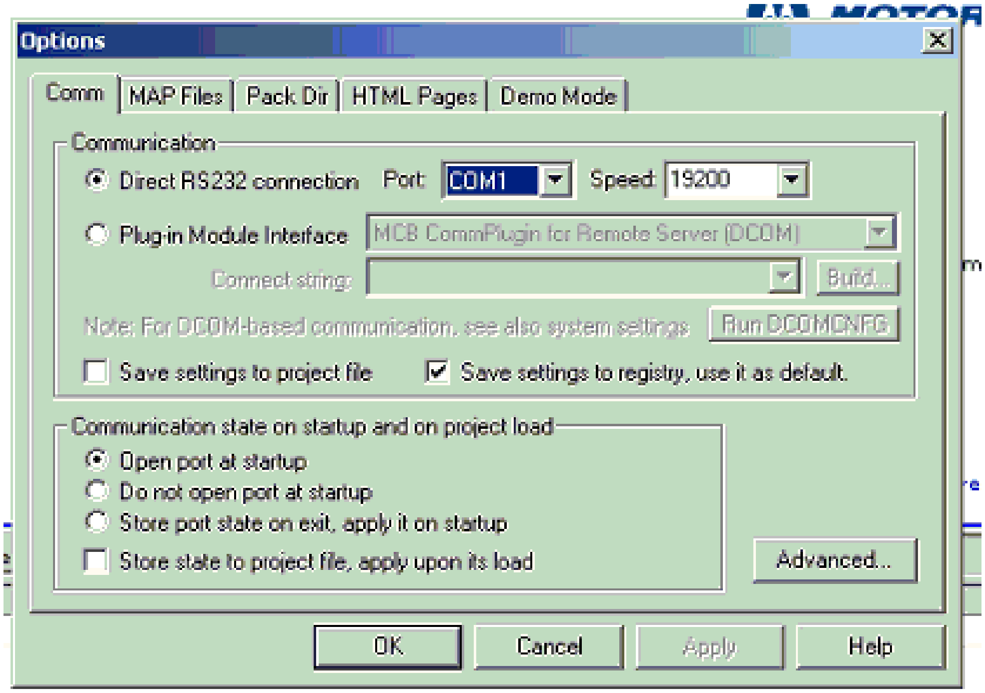
Setup the communication for RS232.

If you are running the PC Master software for the first time, you must setup the communication port and communication speed for the PC. The speed of the demo must be set to 19200 Bauds.

Open *Options* window by selecting the PC Master software menu sequence

Project -> Options

Select the appropriate COMx port and setup the speed to 19200 Baud. See **Figure 2-7**.



Communication

● Direct RS232 connection Port: COM1 Speed: 19200

○ Plug-in Module Interface MCB CommPlugin for Remote Server (DCOM)

Connect string:

Note: For DCOM-based communication, see also system settings

□ Save settings to project file ☑ Save settings to registry, use it as default.

Communication state on startup and on project load

● Open port at startup
○ Do not open port at startup
○ Store port state on exit, apply it on startup
□ Store state to project file, apply upon its load

Figure 2-7. Options window

2.7.3 Step 3

Open the PC Master software project.

You can open a project by selecting the PC Master software menu sequence:

File -> Open Project ->PC Master-BlackBox.pmp

See **Figure 2-8**.

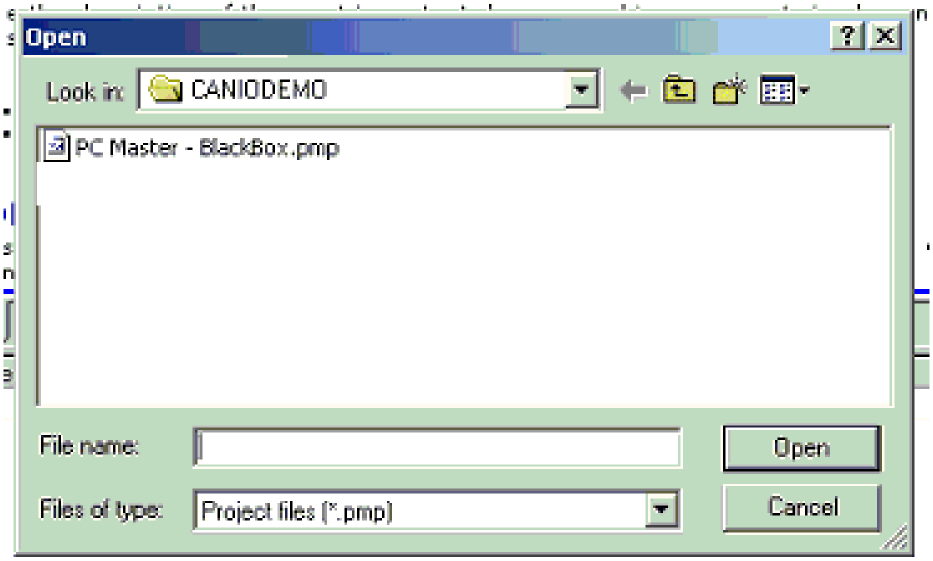


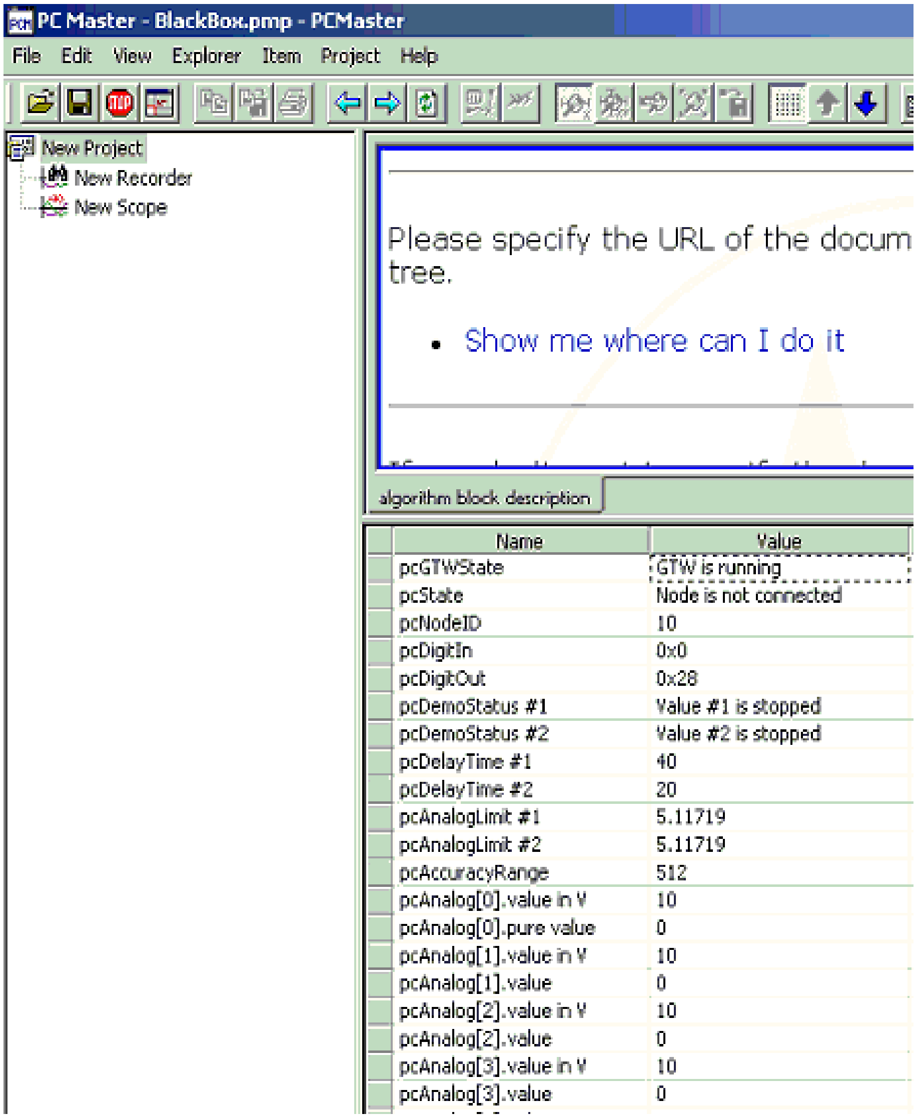
Figure 2-8. Open Project Window

If the project is opened successfully, new information will be displayed on the screen. See **Figure 2-9**. The *Variable Watch* pane contains the list of variables assigned to the watch within the Demo System. The system works as follows: The PC Master software communicates with the PC_CAN Interface application via a well-defined communication protocol. This protocol allows the PC application to issue commands and to read or modify PC_CAN Interface application variables. All commands and variables used in the PC Master software project must be specified within the project.

2.7.4 Step 4

Check that the application is running.

At this phase of the setup process, it is important to pay attention to the variable *pcGTWState* (shown in the Variable Watch pane), to see if the PC_CAN Interface application (gateway) is running. If so, the variable *pcGTWState* must have a value *'GTW is running'*. See **Figure 2-9**.



Please specify the URL of the docum
tree.

- Show me where can I do it

Name	Value
pcGTWState	GTW is running
pcState	Node is not connected
pcNodeID	10
pcDigitIn	0x0
pcDigitOut	0x28
pcDemoStatus #1	Value #1 is stopped
pcDemoStatus #2	Value #2 is stopped
pcDelayTime #1	40
pcDelayTime #2	20
pcAnalogLimit #1	5.11719
pcAnalogLimit #2	5.11719
pcAccuracyRange	512
pcAnalog[0].value in V	10
pcAnalog[0].pure value	0
pcAnalog[1].value in V	10
pcAnalog[1].value	0
pcAnalog[2].value in V	10
pcAnalog[2].value	0
pcAnalog[3].value in V	10
pcAnalog[3].value	0

Figure 2-9. Part of Demo Project PC Master Software Window

2.7.5 Step 5

Select Node ID.

Click on the cell *Value* of the variable *pcNodeID* to select Node ID corresponding with the connected Industrial CAN I/O Module (default setting is to the serial number). If the communication between the PC_CAN Interface and the Industrial CAN I/O Module is OK, the value of the variable *pcState* has to change, from ‘*Node is not connected*’ to ‘*Node is ready*’.

2.7.6 Step 6

Start the PC Master software Oscilloscope.

Click on the item *New Scope* in the *Project Tree* pane to start the Oscilloscope.

PC Master software Oscilloscope is similar to the classical hardware oscilloscope. It graphically shows in real-time the values of selected variables . These values are read from the Board application through the serial communication line. The sampling speed is limited by the communication data rate between the PC Master software and the target Board application (PC_CAN Interface). The data rate of the demo system is 19200bps.

The variables *pcAnalog[0].value in V* and *pcAnalog[1].value in V* are displayed on the scope.

2.7.7 Step 7

Start the control process

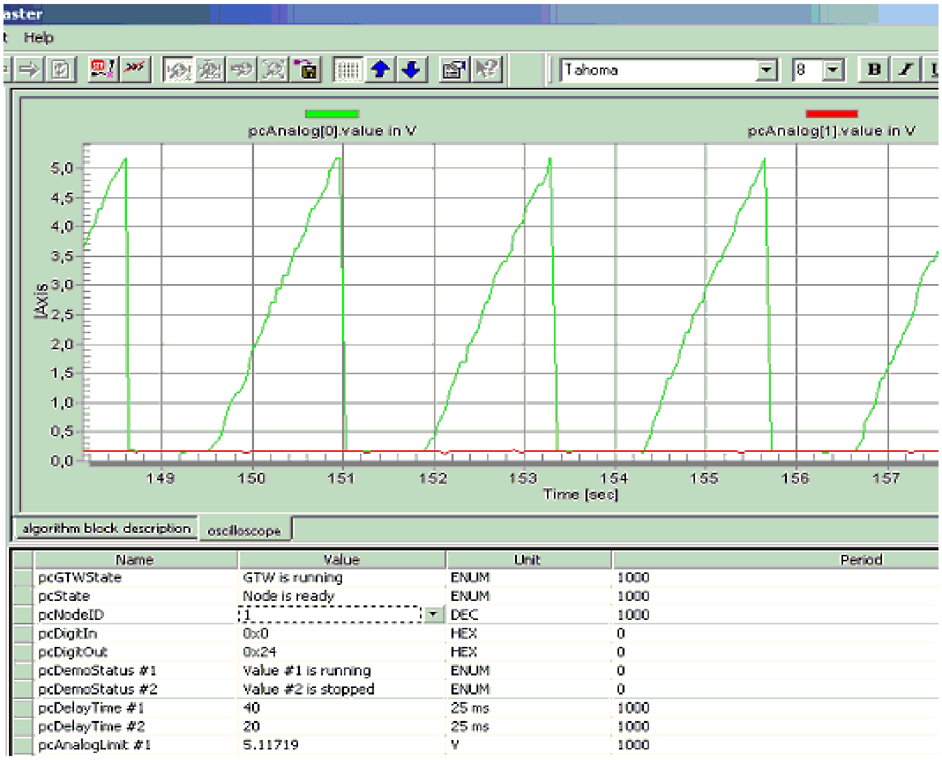
Press the *START* button for channel 1, channel 2, or both, on the Black Box to start the control process. If the button is pressed, its green LED will flash.

You can observe the demo performance on the scope.

See **Figure 2-10**.

The status of each channel control process is indicated by the variables *pcDemoStatus #1* and *pcDemoStatus #2.* Where the process has stopped, the variable has the value ‘*Value #x is stopped’,* and where the process is running, the variable has the value *‘Value #x is running’* (x represents the channel number).

Pressing the *STOP* button stops the control process. If this button is pressed, its red LED will flash.



Name	Value	Unit	Period
pcGTWState	GTW is running	ENUM	1000
pcState	Node is ready	ENUM	1000
pcNodeID	1	DEC	1000
pcDigitIn	0x0	HEX	0
pcDigitOut	0x24	HEX	0
pcDemoStatus #1	Value #1 is running	ENUM	0
pcDemoStatus #2	Value #2 is stopped	ENUM	0
pcDelayTime #1	40	25 ms	1000
pcDelayTime #2	20	25 ms	1000
pcAnalogLimit #1	5.11719	V	1000

Figure 2-10. Oscilloscope Window

2.8 Demo System Variables Description

The *Variable Watch* pane in the bottom of the application window contains the list of watch variables. The selection of watch variables and their graphical properties is defined separately for each project block. When defining a variable, its name, unit, and number format are specified.

Read-only variables can only be monitored. Variables that are defined as modifiable can be altered in the *Variable Watch* pane. For details, refer to *PC Master Software User Manual* Section 4.2.

This section describes the variables defined for the Industrial CAN I/O Demo System.

2.8.1 pcGTWState

Read-only variable. The variable *pcGTWState* shows the status of the application running on the PC_CAN Interface (gateway). The valid values for this variable are

GTW is running

CAN failure

Temperature pre-warning

2.8.2 pcState

Read-only variable. The variable *pcState* shows the status of the connection between the PC_CAN Interface and the Industrial CAN I/O Module. The valid values for this variable are

Node is not connected

Node is ready

2.8.3 pcNodeID

Modifiable variable. The variable *pcNodeID* contains the Node ID of the connected Industrial CAN I/O Module. The valid values for this variable are within the range of 1 to 63.

2.8.4 pcDigitIn

Read-only variable. The variable *pcDigitIn* shows the status of the Industrial CAN I/O Module digital inputs. The valid values for this variable are within the range of 0x0000 to 0xFFFF.

2.8.5 pcDigitOut

Modifiable variable. The variable *pcDigitOut* contains data for the digital outputs setting. The valid values for this variable are within the range of 0x0000 to 0xFFFF.

2.8.6 pcDemoStatus #x

Read-only variable. The variable *pcDemoStatus #x* (x specifies which Black Box channel) shows the status of the control process on that channel. The valid values for this variable are

Value #x is stopped

Value #x is running

2.8.7 pcDelayTime #x

Modifiable variable. The variable *pcDelayTime #x* contains the delay time of the selected channel. For an explanation of delay time see **Figure 2-11**. The valid values for this variable are within the range of 0 to 65535. The actual real delay time is 25ms multiplied by the *pcDelayTime #x* value.

2.8.8 pcAnalogLimit #x

Modifiable variable. The variable *pcAnalogLimit #x* contains the analog limit value for the selected channel. For explanation of the analog limit see **Figure 2-11**. The valid values for this variable are within the range of 0 to 9.

2.8.9 pcAccuracyRange

Read-only variable. The variable *pcAccuracyRange* gives information on the analog channels scaling. The valid values for this variable are 512 (if 10 bit A/D conversion accuracy is selected) or 128 (if 8 bit A/D conversion accuracy is selected).

2.8.10 pcAnalog[x].value in V

Read-only variable. The variable *pcAnalog[x].value in V* (x specifies which Module analog channel) gives the analog value measured on a channel, expressed in Volts. The valid values for this variable are from -10V to 10 V.

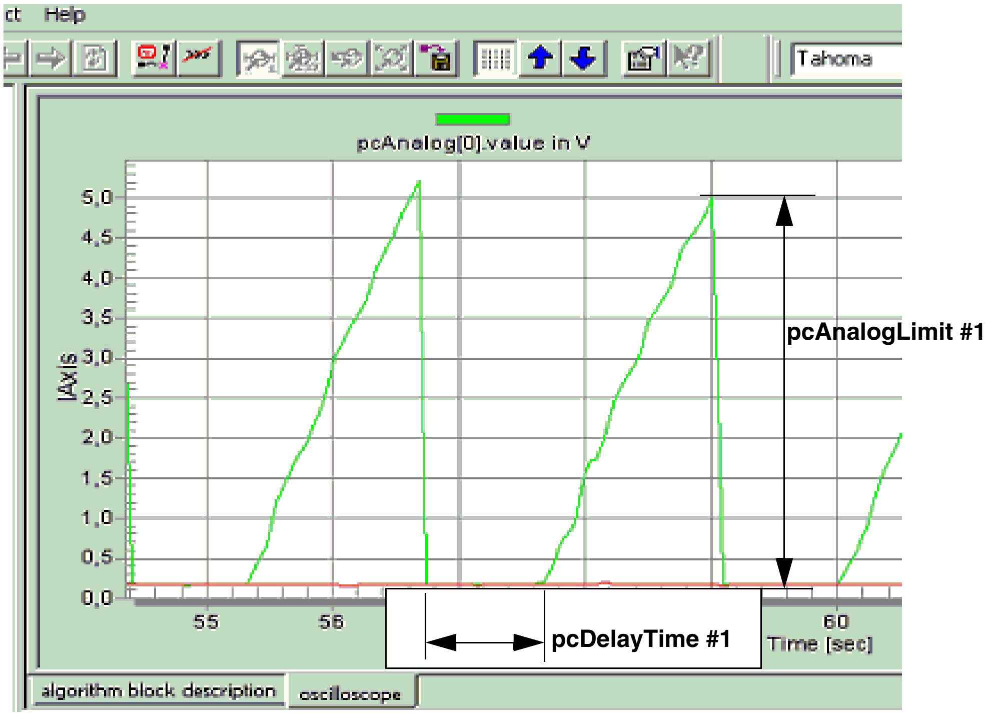


Figure 2-11. Demo System Parameters

2.8.11 pcAnalog[x].value

Read-only variable. The variable *pcAnalog[x].value* (x specifies which Module analog channel) gives the analog value measured on a channel, expressed in digits. The valid values for this variable are from 0 to 1023.

2.8.12 pcAnalog[x].mode

Read-only variable. The variable *pcAnalog[x].mode* gives information on the analog channel mode. The valid values of this variable are

Current Mode

Voltage Mode

2.8.13 pcAnalogConf[x].range

Modifiable variable. The variable *pcAnalogConf[x].range* contains the range of the input amplifier for the selected channel.

NOTE: This feature is not supported in the demo version.

2.8.14 pcAnalogConf[x].accuracy

Modifiable variable. The variable *pcAnalogConf[x].accuracy* contains data on the accuracy of the analog to digital conversion. The data is shred by all analog channels. The valid values for this variable are

8 bit accuracy

10 bit accuracy

Section 3. Hardware Description

3.1 Contents

3.2 Introduction

This reference design of the Industrial CAN I/O Module Reference Design provides an Input/ Output module for industrial automation purpose. Besides this, the Industrial CAN I/O Module Reference Design can be used as a hardware platform for high level communication protocol software development. In addition that, the Industrial CAN I/O Module Reference Design enables the implementation and testing of user software. For this purpose, the board is equipped with a Background Debug Mode (BDM) interface for reprogramming and debugging. The module contains analog inputs, digital inputs, digital outputs, a controller with a CAN transceiver, and optional RS232, RS485 transceivers. The CAN interface is compatible to ISO 11898 and allows a maximum data transfer rate of 500 kbit/s. The block diagram of the module can be seen in **Figure 3-1**.

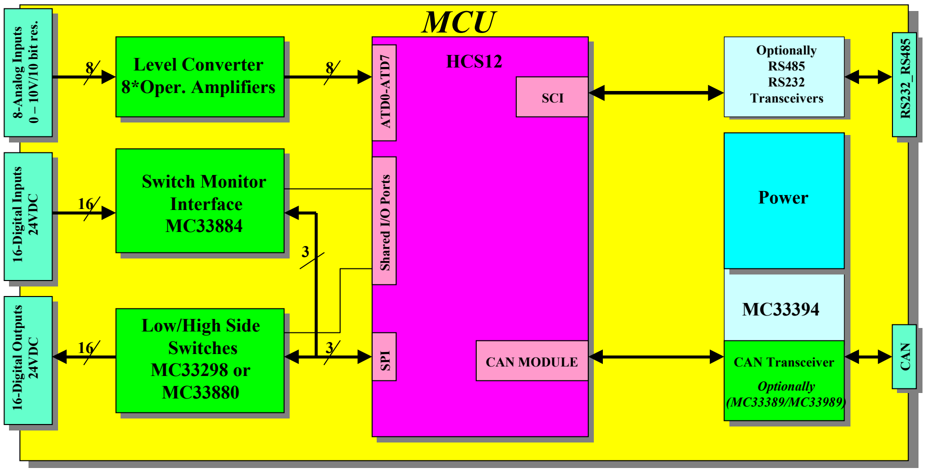


Figure 3-1. Industrial CAN I/O Module Block Diagram

The reference design is based on available analog devices portfolio to accomodate digital inputs (MC33884), outputs (MC33298) and CAN transceiver (PC33394 or MC33388). Analog inputs are based on third party operational amplifiers. The application is supported by HCS12 MCU.

3.3 Technical Data

This section provides technical data for both the Industrial CAN I/O Module Reference Design itself, as well as for the individual Motorola devices used in the Module.

3.3.1 Industrial CAN I/O Module Reference Design

- Analog Inputs
  - Input Ranges

    +/- 10V

    +/- 5V

    0 to 10V

    0 to 5V

4 to 20 mA

- Resolution

  8 bit or 10 bit (programmable)

- Digital Inputs
  - Programmable inputs to monitor

    Switch-to-GND or Switch-to-Power Supply
  - Threshold Voltage 4V
  - Input Voltage Range from –14V to 40V
  - Programmable wetting current (~14mA) pulse to 'clean' oxides from the switch contacts
  - ESD Voltage – Human Body Model (4000V), Machine Model (200V)
- Digital Outputs
  - 3A Peak Current Outputs (RDS(on) of 0.45 Ohm)
  - Output ON and OFF Open Load Detection
  - Internal Clamps to 65V for Inductive Fly-Back Protection
  - Over Current Detection, Shutdown, with Auto-Retry
  - Over and Undervoltage Detection, Shutdown, with Auto-Retry
  - Shorts-to-Ground and Supply Detection, Shutdown, with Auto-Retry
  - Over Temperature Detection, Shutdown, with Auto-Retry
  - Fault Diagnostics Reporting
  - Output Current Limiting
  - ESD Voltage – Human Body Model (2000V), Machine Model (200V)
- CAN Interface
  - Compatible to ISO 11898
  - Programable speed (set up by coding switch)

    125kbps, 250kbps and 500kbps

- Power Supply
  - 24V/190mA

3.3.2 MC9S12DP256 Processor

The control unit of the Industrial CAN I/O Module Reference Design is the MC9S12DP256 microcontroller unit (MCU). It is a 16-bit device composed of STAR12 CPU processing unit and standard on-chip peripherals.

System resource mapping, clock generation, interrupt control and bus interfacing are managed by the System Integration Module (SIM). The MC9S12DP256 has full 16-bit data paths throughout. However, the external bus can also operate in an 8-bit narrow mode, allowing an interface 8-bit wide memory for lower cost systems. The inclusion of a PLL circuit allows power consumption and performance to be adjusted to suit the operational requirements.

3.3.2.1 Features

- 16-bit STAR12 CPU
  - Upward compatible with M68HC11 instruction set
  - Interrupt stacking and programmer's model identical to M68HC11
  - 20-bit ALU
  - Instruction pipe
  - Enhanced indexed addressing
- Multiplexed External Bus
- Memory
  - 256K byte Flash EEPROM
  - 4.0K byte EEPROM
  - 12.0K byte RAM
- Two 8 channel Analog-to-Digital Converters

  - 10-bit resolution
- Five 1M bit per second, CAN 2.0 A, B software compatible modules
  - Four receive and three transmit buffers
  - Flexible identifier filter, programmable as 2 x 32 bit, 4 x 16 bit or 8 x 8 bit
  - Four separate interrupt channels for Rx, Tx, error and wake-up
  - Low-pass filter wake-up function
  - Loop-back for self test operation
  - Time-stamping capabilities for network synchronization
- 8 channel IC/OC Enhanced Capture Timer
- Byte Data Link Controller (BDLC)
- Inter-IC Bus (IIC)
- 8 PWM channels with programmable period and duty cycle
  - Standard 8-bit 8-channel, 16-bit 4-channel, or any combination of 8/16 bit
  - Separate control for each pulse width and duty cycle
  - Left-aligned or center-aligned outputs
  - Programmable clock select logic, with a wide range of frequencies
  - Fast emergency shutdown input
  - Usable as interrupt inputs
- Serial interfaces
  - Two asynchronous Serial Communications Interfaces (SCI)
  - Three synchronous Serial Peripheral Interfaces (SPI)
- SIM (System integration module)
  - CRG (low current oscillator, PLL, reset, clocks, COP watchdog, real time interrupt, clock monitor)
  - MEBI (Multiplexed External Bus Interface)

  - MMC (Module Mapping Control)
  - INT (Interrupt control)
  - BKP (Breakpoints)
  - BDM (Background Debug Mode)
- 112-Pin LQFP package or 80-Pin QFP package
  - 50 MHz CPU equivalent to 25MHz bus operation
  - 2.25 to 2.75V Digital Supply Voltage, generated using an internal voltage regulator
  - 4.75V to 5.25V Analog and I/O Supply Voltage

3.3.3 MC33884 Switch Monitor Interface

The MC33884 provides ON/OFF status reporting of multiple external system switches. The device efficiently interfaces between system electrical switches and low voltage microprocessors. All inputs are protected against transients when implemented, with recommended discharge capacitors placed on the inputs.

The MC33884 can run in four operational modes:

- Sleep mode
  - This mode reduces current to 10 µA and disables the devices.
- Normal mode
  - The device interrupts the µP when an external switch status changes.
- Polling mode
  - The device periodically reads a switch status and interrupts the µP only when a switch is “closed”, reverting the operation back to Normal mode.
- Polling + INT Timer mode
  - The device interrupts the µP when a switch is sensed as “closed”, or when an internal timer "times out". Mode continues with all switches “open”, otherwise reverts back to Normal mode.

3.3.3.1 Features

- Full Operation with 7.0 V ≤ VPWR ≤ 26 V, Limited Operation with 5.5 V ≤ VPWR ≤ 7.0 V
- Input Voltage Range: –14 V to 40 V
- Interfaces Directly to Microprocessors Using SPI Protocol
- 24–Lead Wide Body SOIC Package
- Wake–up on Change in Monitored Switch Status
- Programmable Wetting Current
- 4 Programmable Inputs to Monitor, 4 Switch–to–Battery or 4 Switch–to–Ground Switches
- 6 (fixed function) Inputs to Monitor, 6 Switch–to–Ground Switches
- 2 (fixed function) Inputs to Monitor, 2 Switch–to–Battery Switches
- Standby Current During Normal Mode = 100 uA
- Quiescent Current in Sleep Mode < 10 uA
- Reset (RST) Input Defaults the Device to Sleep Mode
- Active Interrupt (INT) on Change of Switch State in Normal Mode
- 4 Modes of Operation (Sleep, Normal, Polling, Polling + INT Timer)
- Designed to Operate –40°C ≤ TA ≤ 105°C

3.3.4 MC33298 Octal Serial Switch

The MC33298 is an eight output low side power switch, with an 8 bit serial input control. The device interfaces directly with a microcontroller to control various inductive or incandescent loads. The circuit's innovative monitoring and protection features are: very low standby current, cascadable fault reporting, internal 65 V clamp on each output, output specific diagnostics, and independent shutdown of outputs.

3.3.4.1 Features

- Designed to Operate Over Supply Voltages of 5.5 V to 26.5 V

- Interfaces Directly to Microprocessor Using SPI Protocol
- SPI Communication for Control and Fault Reporting
- 8–Bit Serial I/O is CMOS Compatible
- 3.0 A Peak Current Outputs with Maximum RDS(on) of 0.45 W at 25°C
- Outputs are Current Limited to 3.0 A to 6.0 A
- Output Voltages Clamped to 65 V During Inductive Switching
- Maximum Sleep Current (IPWR) of 50 uA with VDD ≤ 2.0 V
- Maximum of 4.0 mA IDD During Operation
- Maximum of 2.0 mA IPWR During Operation with All Outputs “On”
- Open Load Detection (Outputs “Off”)
- Overvoltage Detection and Shutdown
- Each Output has Independent Over Temperature Detection and Shutdown
- Output Mode Programmable for Sustained Current Limit or Shutdown
- Short Circuit Detect and Shutdown with Automatic Retry for Every Write Cycle
- Serial Operation Guaranteed to 2.0 MHz

3.3.5 MC33394 SMPS & CAN Transceiver

The PC33394 is a multi-output power supply integrated circuit with a high speed CAN transceiver. The device incorporates a switching pre-regulator operating over a wide input voltage range from +4.0V to +26.5V (with transients up to 45V).

The switching regulator has an internal 3.0A current limit, and runs in both buck mode or boost mode, to always supply a pre-regulated output followed by Low Drop Out (LDO) regulators.

Additional features include Active Reset circuitry, watching VDDH, VDD3_3, VDDL and VKAM, and user selectable Hardware Reset Timer (HRT).

Power Sequencing circuitry guarantees the core supply voltages never exceed their limits or polarities during system power up and power down.

A high speed CAN transceiver physical layer interfaces between the microcontroller CMOS outputs and the differential bus lines. The CAN driver is short circuit protected, and tolerant of loss of battery or ground conditions.

3.3.5.1 Features

- Wide operating input voltage range: +4.0V to +26.5V (+45V transient).
- Provides all regulated voltages for MCUs and other ECU’s logic and analog functions.
  - VDDH / 5.0V @ 400mA
  - VDD3_3 / 3.3V @ 120mA
  - VDDL / 2.6V (User scalable between 3.3V – 1.25V) @ 400mA
  - The Keep Alive regulator VKAM (scalable to track VDDL) @ 60mA
  - FLASH memory programming voltage VPP / 5.0V or 3.3V @ 150mA
  - The sensor supply outputs VREF(1,2,3) / 5.0V (tracking VDDH) @ 100mA each
- Accurate power up/down sequencing.
- Provides necessary MCU support monitoring and fail–safe support.
- Provides three 5.0 V buffer supplies for internal & external (short–circuit protected) sensors.
- Includes step–down/step–up switching regulator, to provide supply voltages during different battery conditions.

- Interfaces Directly to Standard 5.0V I/O for CMOS Microprocessors by means of Serial Peripheral Interface.

3.3.6 MC33388 CAN Interface

The MC33388 is a Fault Tolerant CAN physical interface device. It operates in differential mode, allowing ground shifts up to 1.5V, reducing RFI disturbances. It offers very low standby current in sleep and standby mode operation, and supports communication speeds up to 125kBauds.

It is fully protected against harsh environments, and the driver is able to detect fault conditions, automatically switching into the appropriate default mode. Under fault condition, it continuously monitors bus failures, in order to switch the bus operation back to normal as soon as the faults disappear.

3.3.6.1 Features

- Very low sleep/standby current (15mA typical)
- Baud rate from 10 kBaud up to 125kBauds
- Automatic switching to single wire mode in the case of bus failure, and return to differential mode if bus failure disappears
- Supports one wire transmission modes with ground offset up to 1.5V
- Internal bus driver slope control function to minimize RFI
- Bus line short-circuit protected to battery, VDD and ground
- Bus line protected against transients
- Thermal protection of bus line drivers
- Supports unshielded twisted pair bus
- An unpowered node does not disturb the bus lines
- Wake-up capability triggered from bus message and wake-up input pin
- Wake-up pin with dual edges sensitivity
- Battery fail flag reported on NERR output

- Ambient temperature range from -40°C to 125°C.

3.3.7 Industrial CAN I/O Module Reference Design Functionality

The Industrial CAN I/O Module Reference Design is dedicated for use in the Industrial Automation market.

3.4 Industrial CAN I/O Module Reference Design Architecture

Schematics of the Industrial CAN I/O Module Reference Design are provided in **Appendix B. Bill of Materials and Schematics**. The Industrial CAN I/O Module Reference Design block diagram can be seen in **Figure B-1**.

The Industrial CAN I/O Module Reference Design is a modular system, designed to demonstrate the performance of Motorola analog devices, as well as the communication capability of the CAN bus in the Industrial environment.

The Industrial CAN I/O Module Reference Design is logically divided into the following three basic blocks:

- Base Board
- Power Supply
- I/O Board

Data transfer between the boards is ensured by the SPI protocol.

3.4.1 The Base Board

The main function of this part of the Industrial CAN I/O Module Reference Design is to control the module and communicate with the system control unit. Eight analog channels are built into this board.
The Base Board block diagram can be seen in **Figure B-2**.
This board is logically divided into the following four basic blocks:

- Microcontroller
- CAN Interface

- Analog Inputs
- RS232_485 Interface

3.4.1.1 Microcontroller

A Motorola 16-bit MC9S12DP256 microcontroller unit (MCU) is the main component (U26) of the Base Board. You can find more details about the microcontroller in section **3.3.2**. The schematic diagram can be seen in **Figure B-3**.

The Industrial CAN I/O Module Reference Design uses one of five CAN modules (CAN0) built into the MCU. The module is a communication controller implementing the CAN 2.0 A/B protocol as defined in the BOSCH specification. It is the specific implementation of the Motorola Scalable CAN (MSCAN). The MSCAN uses 2 external pins, 1 input (RxCAN0) and 1 output (TxCAN0). The signals, named SS_CAN and EN, are assigned to the CAN transceiver chip control.

The SPI module allows full-duplex, synchronous, and serial communication between the MCU and peripheral devices. The Industrial CAN I/O Module Reference Design uses one of three SPI modules (SPI0) built into the MCU. When the SPI system is enabled, the four associated SPI port pins are dedicated to the SPI function as:

- Serial clock (SCK)
- Master out/slave in (MOSI)
- Master in/slave out (MISO)
- Slave select (see NOTE)

The SPI can be configured to operate as a master or as a slave. The master mode must be selected for the SPI0 module, because only a master SPI module can initiate transmissions.

NOTE: *During an SPI transmission, data is transmitted (shifted out serially) and*

received (shifted in serially) simultaneously. The serial clock (SCK) synchronizes shifting and sampling of the information on the two serial data lines (MOSI, MISO). A slave select line allows selection of an individual slave SPI device; slave devices that are not selected do not *interfere with SPI bus activities. The MCU uses the general-purpose I/O port pins for the selection of an individual slave SPI device. The names of the select lines are CSB0, CSB1, CSB2, CSB3, CSB4.*

The Industrial CAN I/O Module Reference Design uses one of two 8 channel Analog-to-Digital (A/D) Converters (AN0) built into the MCU. The A/D module performs analog to digital conversions. This module contains all the necessary analog and digital electronics to perform a single analog to digital conversion. The resolution of the A/D converter is program selectable, at either 8 or 10 bits. It is capable of accepting 5 volts input without permanent damage while simultaneously operating at 5 volts. For power, this module requires VDD, VDDA, VSS, VSSA.

The board provides an 8-position DIP switch (SW1), for configuring the Node ID as well as the CAN speed. The Node ID setup signals (ID0 - ID5) occupy port K, pins PK0 to PK5. The CAN speed setup signals (BDR0,BDR1) are connected to port P, pins PP0 and PP1. The weight meaning of the SW1 positions is determined by software. For more details see sections **2.6.1** and **2.6.2**.

The ports A, B, H, T of the MCU are used as a general-purpose I/O, to drive and sense analog channels settings.

Single-wire communication with host development system is done through the Background Debug Mode (BDM) system, implemented in on-chip hardware. Connection to the target system is made through the standard six pins BDM Connector (J8). For details see section **3.6.8**.

3.4.1.2 The CAN Interface

Each CAN station is connected physically to the CAN bus lines through a transceiver chip. The transceiver is capable of driving the large current needed for the CAN bus and has current protection against defected CAN or defected stations. There are two possible options for the CAN transceiver on the board. One of them is to use Fault Tolerant CAN Interface MC33388; the other one is to use the high speed CAN transceiver part of the MC33394 device, connected through CAN connector J4. Detailed description of the J4 CAN connector can be found in section **3.6.3**. The standard option is to use the CAN transceiver part of the MC33394 device, as MC33388 is not populated on the board.

The MC33388 was designed on the board mainly for development purposes. The schematic diagram of the MC33388 can be seen in **Figure B-4**, and of the MC33394 in **Figure B-15**.

3.4.1.3 The Analog Inputs

The module provides eight analog channels. The blocks diagram of the analog input channels can be seen in **Figure B-6**.

The analog inverter U25A creates a negative reference voltage, common to all analog channels.

The schematic of the individual analog channels can be seen in **Figure B-7** to **Figure B-14** The following description is for analog channel #0 (see **Figure B-7**).

There is a passive low-pass filter with an attenuation slope of -40dB/dec, and with a cut off frequency of 1kHz, in the input of each analog channel.

This input filter is used to eliminate unwanted high-frequency noise and interference, introduced prior to sampling. The filter is created by resistors R2, R3 and capacitors C2, C3.

The dual diode D1 serves to clamp the voltage level applied to the input amplifier to the power supply range of the device.

The resistor R6 (250 ohm, 1/2 watt precision resistor) provides current sensing, as well as substantial over-current-protection, for 4-20mA current loops. The CSI signal indicates if the analog channel is in Voltage (CSI is high) or Current Sensing (CSI is low) mode.

Next, a two stage amplifier, created by the operational amplifiers U1A, U1B and electronic switch U2, adapts the input analog signal to the range 0 to 5V, to meet the input range of the A/D converter. The dual diode, D2, serves to clamp the voltage level applied to the microcontroller input to the power supply range of the device. The analog channel provides a voltage input signal in the range 0 to 10V, 0 to 5V, -5V to +5V, -2.5V to +2.5V, or a current in the range 4 to 20 mA. The input range is controlled by the signals G, G/2 and solder jumper SM1. More details can be found in the **Table 3-1**.

Table 3-1. Input Range Control Signals

Input signal range	Control signals		
	G	G/2	SM1
-5V to +5V	0	0	N
0 to 10V	0	1	N
-2.5V to +2.5V	1	0	N
0 to 5V	1	1	N
4 to 20mA	1	1	S

3.4.1.4 The RS232_485 Interface

The Base Board provides an RS-232 interface for connection to a PC, or a similar host, as well as an RS-485 interface that can be used for industrial applications. Refer to the RS232_485 schematic diagram in **Figure B-5**. The RS-232 level converter (U28) transforms the SCI +5V signal levels to RS-232 compatible signal levels, and connects to the host's serial port via connector J6. See block diagram **Figure B-2**. The connector is arranged as a DCE port. Flow control is not provided.

The RS-485 transceiver (U29) is capable of bidirectional data communications on multipoint bus transmission lines. The standard configuration of the RS-485 interface is Full-Duplex mode. Using jumper JP1 the configuration can be changed to Half-Duplex mode. A jumper on pins 1 and 2 connects lines A485 and Y485, and a jumper on pins 3 and 4 connects lines B485 and Z485.

To minimize reflections, the line should be terminated in its characteristic impedance by jumper JP2. A jumper on pins 1 and 2 connects resistor R128 between lines Y485 and Z485, and a jumper on pins 3 and 4 connects resistor R129 between lines A485 and B485.

RS-485 interface is connected to the network via connectors J6 (DCE arrangement) and J7 (DTE arrangement). See block diagram **Figure B-2**.

3.4.2 The Power Supply

The power supply is designed to meet all the power supply needs of Industrial CAN I/O Module. The schematic of the power supply can be seen in **Figure B-15**. The main device of the board is the PC33394, a multi-output power supply integrated circuit with a high speed CAN transceiver. This device incorporates a switching regulator to supply a pre-regulated output, followed by Low Drop Out (LDO) regulators. The module uses VDDH / 5.0V and two sensor supply outputs VREF(#2 and #3) / 5.0V. The PC33394 device is not fully utilized for this application. There is no need for a power supply of 3.3V or less on the module, so supplies VDD3_3 / 3.3V, VDDL / 2.6V, and VPP/5.0V are not used.

The internal switching regulator incorporates circuitry to implement a Buck or a Buck/Boost regulator. Only the Buck regulator is implemented on this board. The flyback converter provides symetrical voltages to supply the analog channels of the module. The output voltage is derived through the transformer T1, rectified (D1, D2), and a symetrical +12V, -12V output voltage is generated by the linear regulators U1,U2.

The PC33394 device opens up three Reset pins:

/PORESET - Power On Reset

/PRERESET - Pre Reset

/HRESET- Hardware Reset

All the Reset pins are open drain ‘active low’ outputs. The module uses /HRESET signal connected to the microcontroller /HRESET pin with a pull-up resistor on the microcontroller side.

A high speed CAN transceiver physical layer interfaces between the microcontroller CMOS outputs and the differential bus lines. The CAN driver is short circuit protected and tolerant of loss of battery or ground conditions.

3.4.3 The I/O Board

The I/O Board provides 16 digital inputs and 16 digital outputs. The block diagram of the I/O Board can be seen in **Figure B-16**. The I/O Board is connected through the connector J1.

3.4.3.1 The Input Block

Two Switch Monitor Interface MC33884 devices on the board provide an interface between electrical switches and microcontroller. The schematic of the Input Block can be seen in **Figure B-18**. The MC33884 monitors the OPEN/CLOSED status of multiple external switches used in the system. The device supplies switch contact pull-up and pull-down current, while monitoring the input voltage level. All inputs are protected for transients with a static discharge capacitor used on the inputs.

The MC33884 device can run in four modes of operation - Sleep, Normal, Polling, and Polling + INT Timer. All modes of operation are programmed via the Serial Peripheral Interface (SPI) control. The response to a SPI command will always return Switch Status, Master/Slave, INT Flag, and Mode settings. More details can be found in the datasheet MC33884/D.

The Module uses 4 programmable Switch-to-Ground or Battery sense inputs, and 4 Switch-to-Ground sense inputs of each MC33884 device. The devices on the board are used in parallel configuration.

The microcontroller selects the MC33884 to be communicated with through the use of the CSB2 or CSB3 pins. With the CSBx in a logic low state, command words may be sent to the selected device.

3.4.3.2 The Output Block

Two eight output low side power switch MC33298 devices on the board interface directly with a microcontroller, to control various inductive or incandescent loads. The schematic of the Output Block can be seen in **Figure B-17**.

The devices on the board are programmed via the SPI control, and are used in parallel configuration. The microcontroller selects the MC33298 to be communicated with through the use of the CSB0 or CSB1 pins. With the CSBx in a logic low state, command words may be sent to the selected device. The response to a SPI command will always return the status of the device's output switches.

The status of the device's output switches is also signalled by LEDs D1 to D16. Flashing LED indicates ON status.

3.5 Boards Layout

Detailed layout plans of the Industrial CAN I/O Module Reference Design boards, with the names of all components are shown in this section.

3.5.1 Base Board Layout

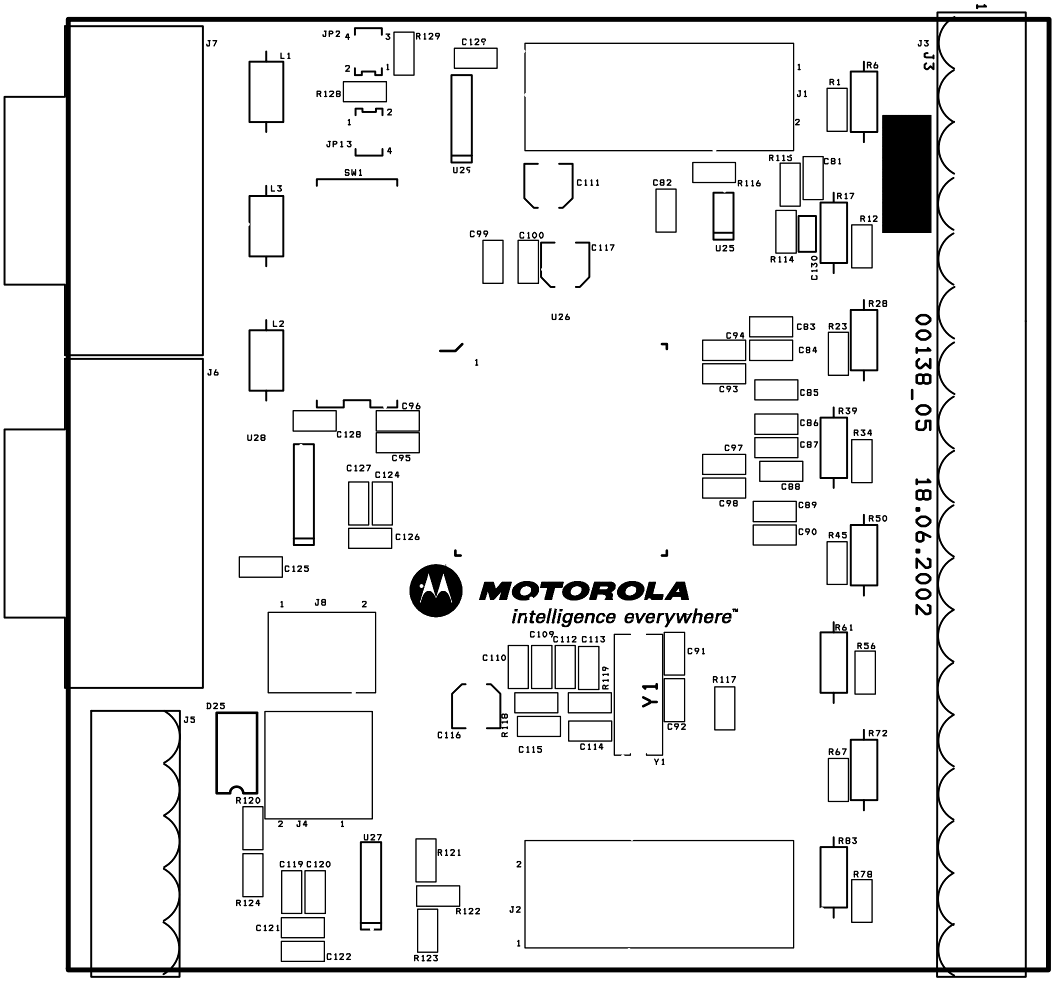


Figure 3-2. Base Board Component Side Layout

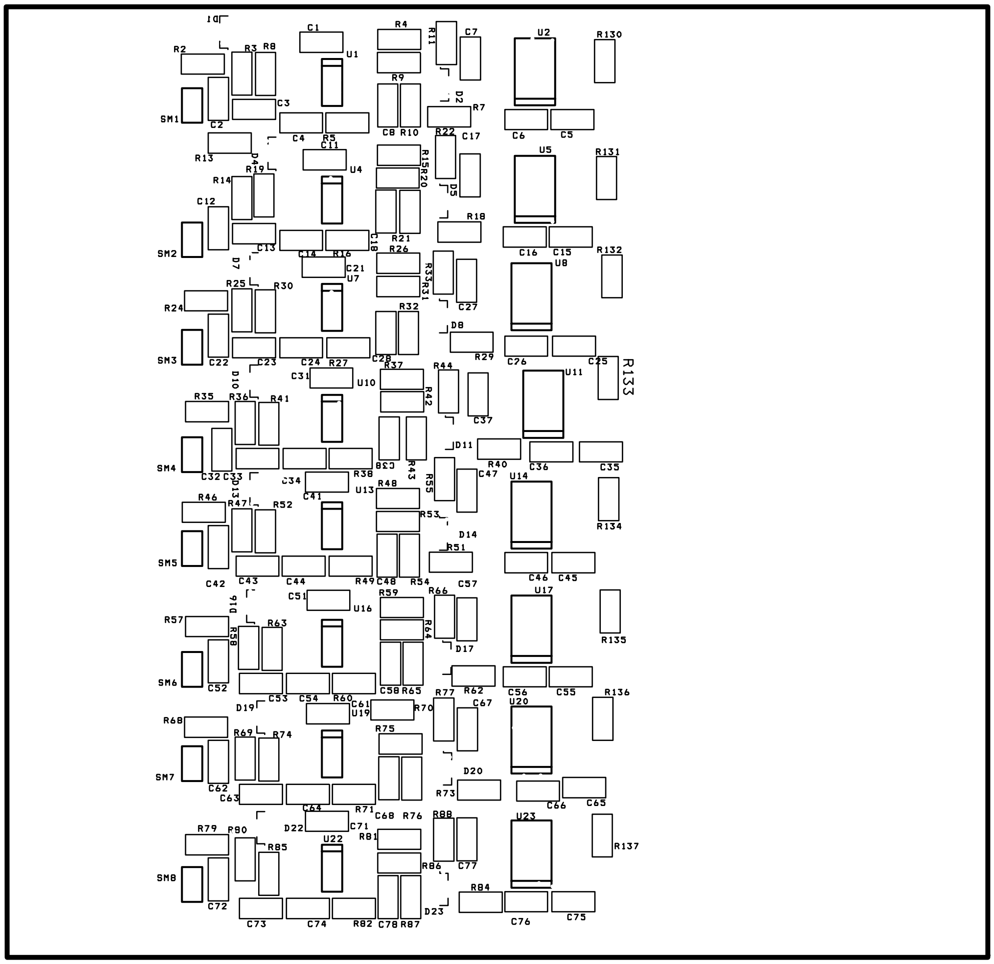


Figure 3-3. Base Board Solder Side Layout

3.5.2 Power Board Layout

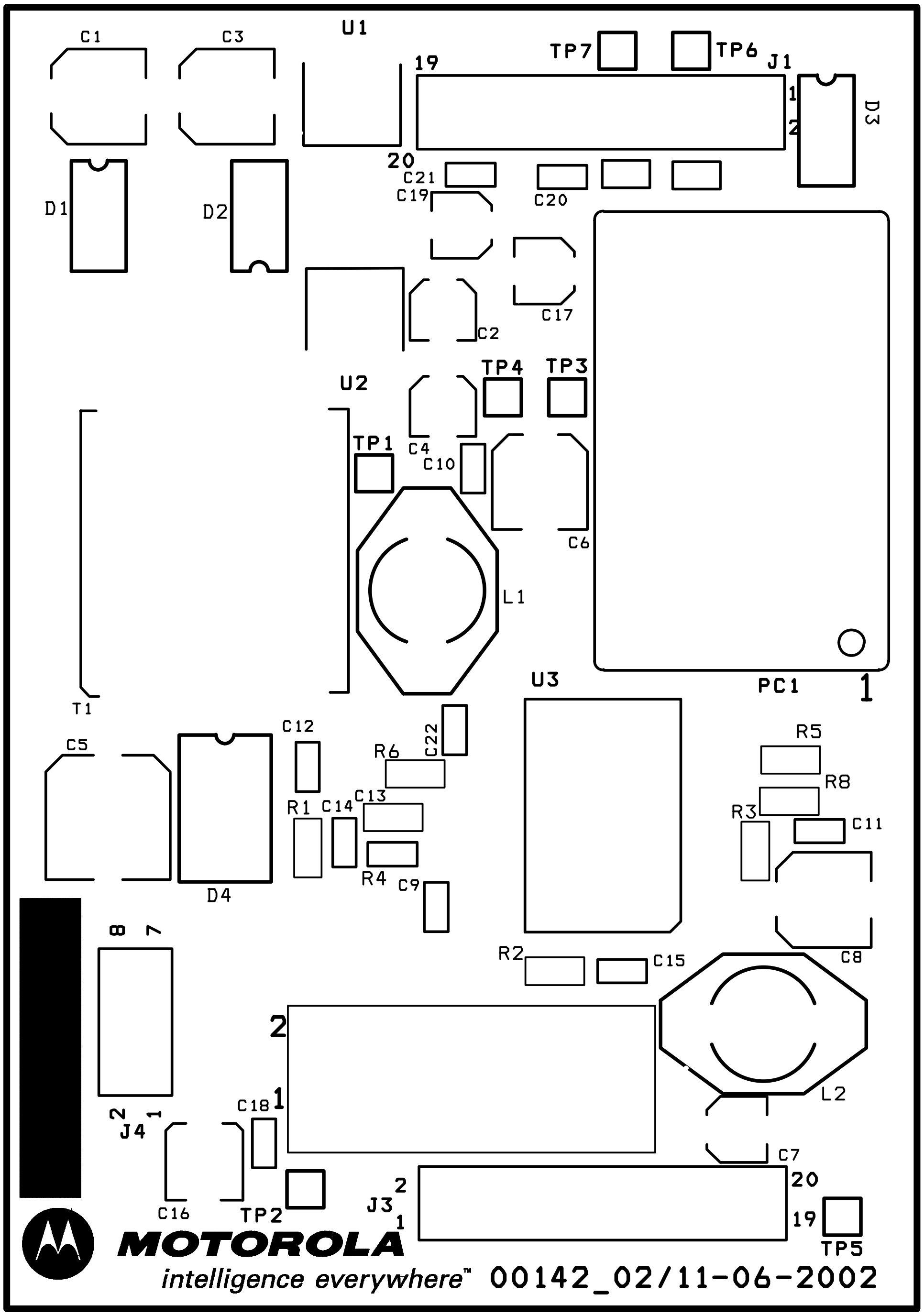


Figure 3-4. Power Board Component Side Layout

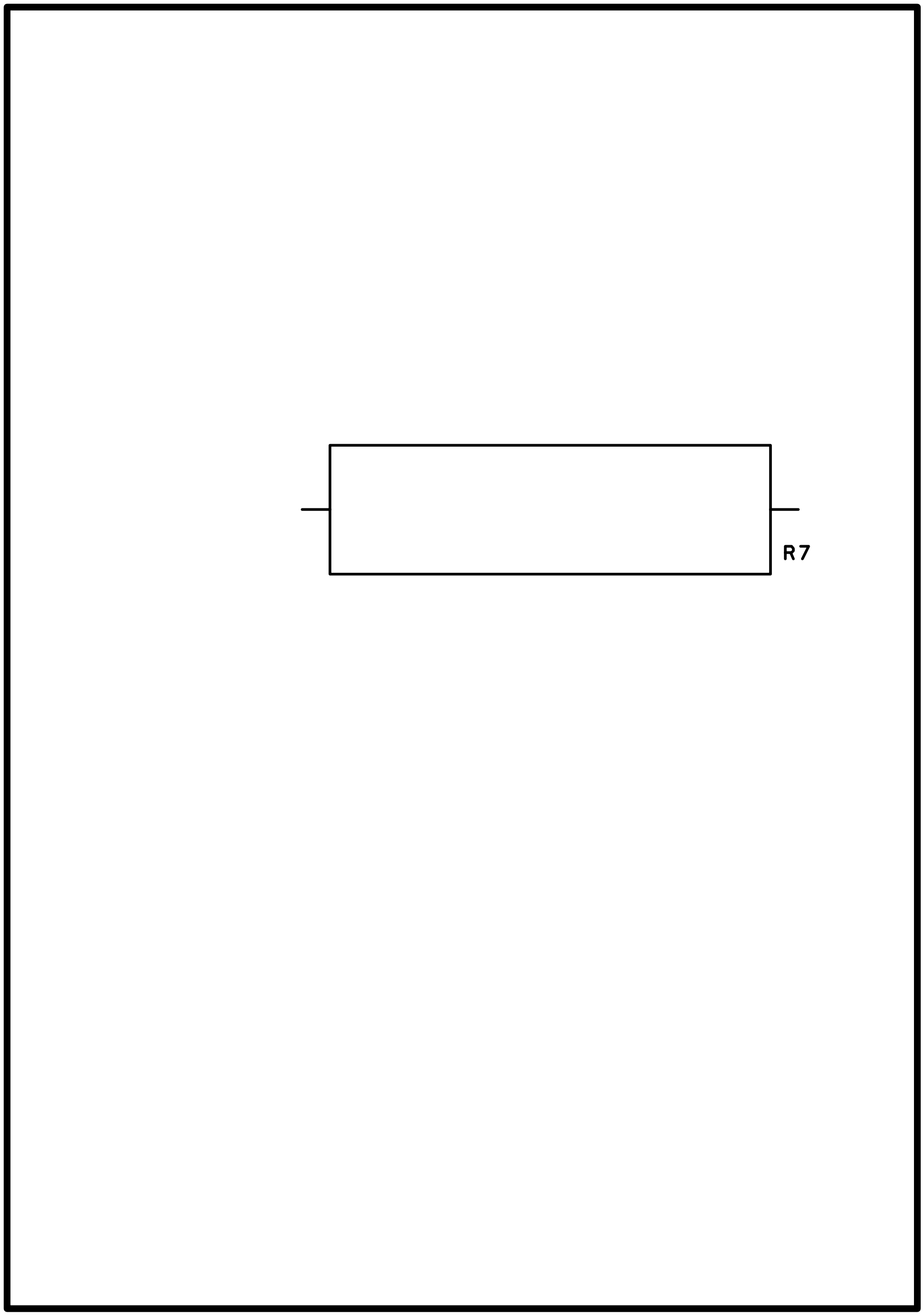


Figure 3-5. Power Board Solder Side Layout

3.5.3 I/O Board Layout

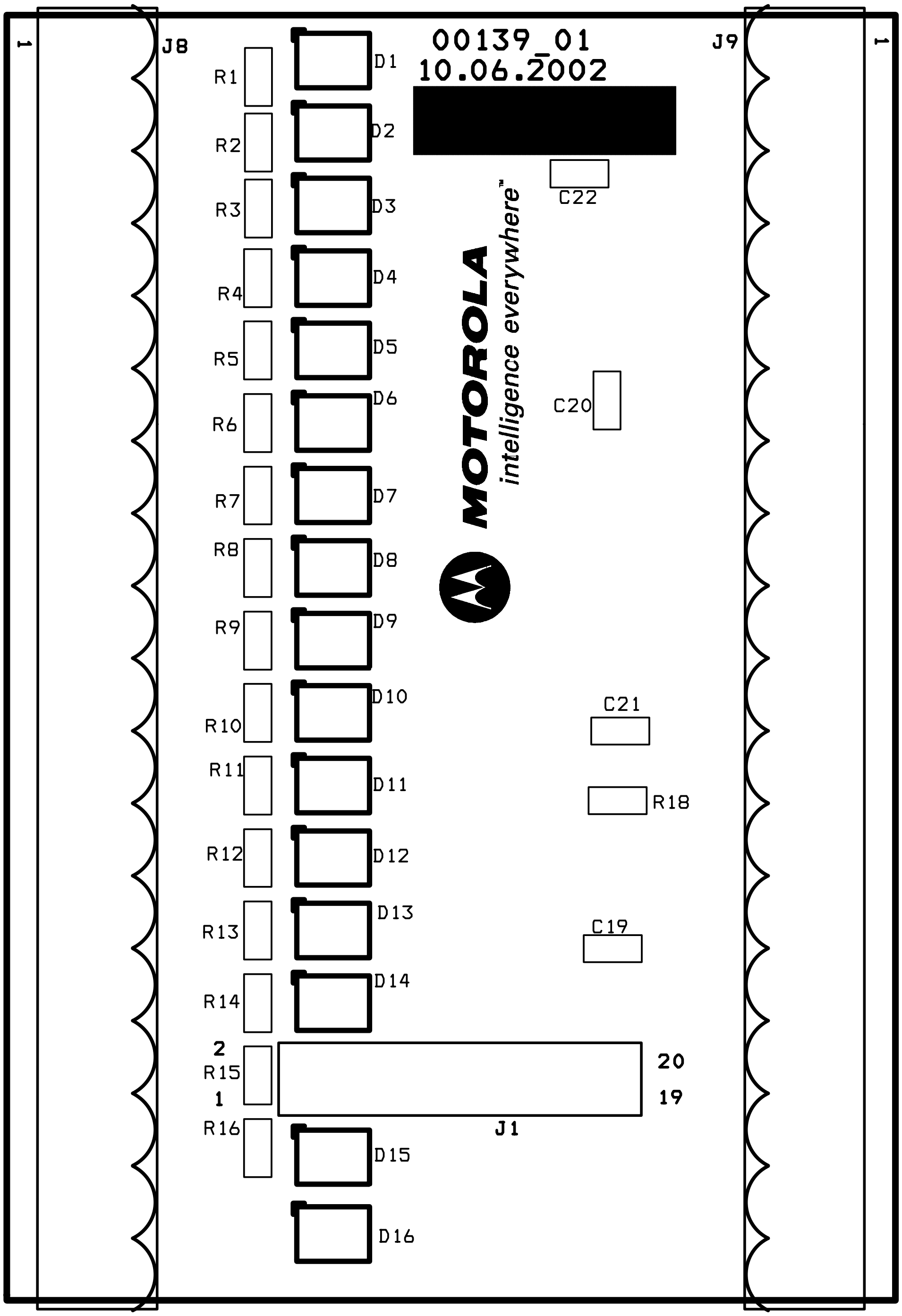


Figure 3-6. I/O Board Solder Side Layout

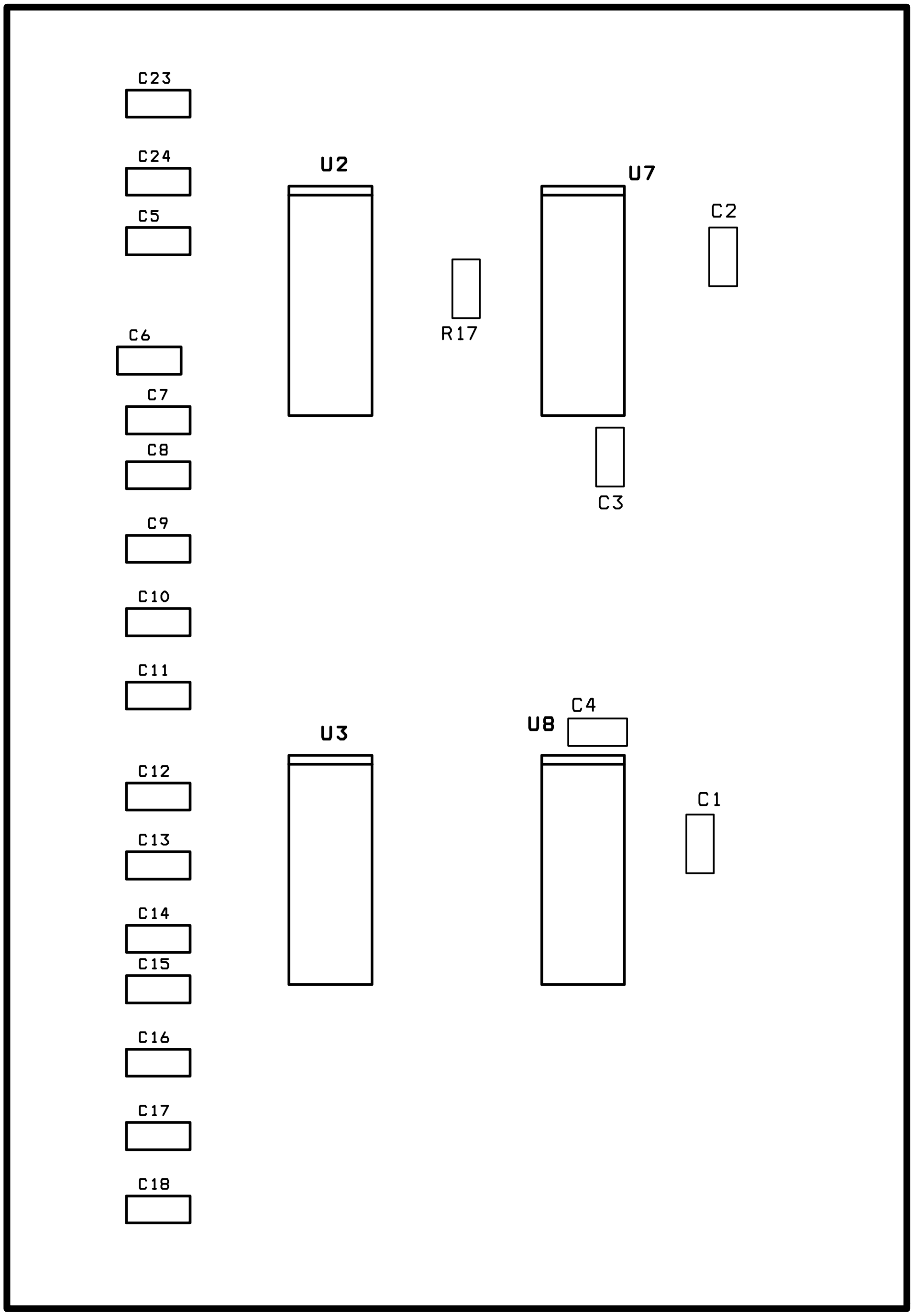


Figure 3-7. I/O Board Solder Side Layout

3.6 Base Board Connectors

This section provides information of the Base Board connectors pin assignment and meaning.

3.6.1 Power Supply Connector - J1

PWR_24	**1**	**2**	PWR_24	
DGND	**3**	**4**	DGND	Digital Ground
+12VA	**5**	**6**	+12VA	+12V voltage for analog block power supply
AGND	**7**	**8**	AGND	Analog Ground
-12VA	**9**	**10**	-12VA	-12V voltage for analog block power supply
A5V	**11**	**12**	A5V	+5V voltage for analog block power supply
GNDA5V	**13**	**14**	GNDA5V	Ground related to A5V
R5V	**15**	**16**	R5V	+5V reference voltage
GNDR	**17**	**18**	GNDR	Ground related to R5V
NC	**19**	**20**	NC	Not Connected

3.6.2 I/O Control Connector - J2

+5V power supply	VDD	**1**	**2**	VDD	+5V power supply
SPI Data In	SI	**3**	**4**	SO	SPI Data Out
SPI Clock	SCLK	**5**	**6**	CSB3	Chip select #3
Chip select #2	CSB2	**7**	**8**	FSPD1	Control signal #1
Chip select #1	CSB1	**9**	**10**	FSPD0	Control signal #0
Reset Output Devices	RESOUT	**11**	**12**	CSB0	Chip select #0
Reset Input Devices	RESIN	**13**	**14**	INTB	Interrupt signal
Microcontroller Reset	/HRESET	**15**	**16**	CSB4	Chip select #4
+24V module power supply	VPWR	**17**	**18**	VPWR	+24V module power supply
Digital Ground	DGND	**19**	**20**	DGND	Digital Ground

3.6.3 CAN Connector - J4

CAN bus drive high line	CANH	**1**	**2**	CANL	CAN bus drive low line
Digital Ground	DGND	**3**	**4**	DGND	Digital Ground
CAN Receive Data	RxCAN	**5**	**6**	TxCAN	CAN Transmit Data
CAN interrupt	INT	**7**	**8**	N.C.	Not Connected

3.6.4 Analog Inputs Connector - J3

1	PWR_24V	
2	DGND	Digital Ground
3	AI0	Analog Input #0
4	AGND0	Analog Return #0
5	AI1	Analog Input #1
6	AGND1	Analog Return #1
7	AI2	Analog Input #2
8	AGND2	Analog Return #2
9	AI3	Analog Input #3
10	AGND3	Analog Return #3
11	AI4	Analog Input #4
12	AGND4	Analog Return #4
13	AI5	Analog Input #5
14	AGND5	Analog Return #5
15	AI6	Analog Input #6
16	AGND6	Analog Return #6
17	AI7	Analog Input #7
18	AGND7	Analog Return #7

3.6.5 Module Control Connector - J5

1	V-	+24V power supply return
2	CANL	CAN bus drive low line
3	NC	Not Connected
4	CANH	CAN bus drive high line
5	V+	+24V power supply

3.6.6 RS232/RS485 Connector - J6

1	A485	RS485-A-Terminal
2	RX	RS232 - RX
3	TX	RS232 - TX
4	xxx	Jumper to 6
5	DGND	Digital Ground
6	B485	RS485-B-Terminal
7	xxx	Jumper to 8
8	Y485	RS485-Y-Terminal
9	Z485	RS485-Z-Terminal

3.6.7 RS485 Connector - J7

1	A485	RS485-A-Terminal
2	NC	Not Connected
3	NC	Not Connected
4	NC	Not Connected
5	DGND	Digital Ground
6	B485	RS485-B-Terminal
7	NC	Not Connected
8	Y485	RS485-Y-Terminal
9	Z485	RS485-Z-Terminal

3.6.8 BDM Connector - J8

Background Interface	BKGD	**1**	**2**	GND	Digital Ground
Not Connected	N.C.	**3**	**4**	RESET\	MCU Reset
Not Connected	N.C	**5**	**6**	VDD	+5V Power supply

3.7 Power Supply Board Connectors

This section provides information of the Power Supply Board connectors pins assignment and meaning.

3.7.1 Power Supply Connector - J1

PWR_24	**1**	**2**	PWR_24	
DGND	**3**	**4**	DGND	Digital Ground
+12VA	**5**	**6**	+12VA	+12V voltage for analog block power supply
AGND	**7**	**8**	AGND	Analog Ground
-12VA	**9**	**10**	-12VA	-12V voltage for analog block power supply
A5V	**11**	**12**	A5V	+5V voltage for analog block power supply
GNDA5V	**13**	**14**	GNDA5V	Ground related to A5V
R5V	**15**	**16**	R5V	+5V reference voltage
GNDR	**17**	**18**	GNDR	Ground related to R5V
NC	**19**	**20**	NC	Not Connected

3.7.2 I/O Control Connector - J2

+5V power supply	VDD	**1**	**2**	VDD	+5V power supply
SPI Data In	SI	**3**	**4**	SO	SPI Data Out
SPI Clock	SCLK	**5**	**6**	CSB3	Chip select #3
Chip select #2	CSB2	**7**	**8**	FSPD1	Control signal #1
Chip select #1	CSB1	**9**	**10**	FSPD0	Control signal #0
Reset Output Devices	RESOUT	**11**	**12**	CSB0	Chip select #0
Reset Input Devices	RESIN	**13**	**14**	INTB	Interrupt signal
Microcontroller Reset	/HRESET	**15**	**16**	CSB4	Chip select #4
+24V module power supply	VPWR	**17**	**18**	VPWR	+24V module power supply
Digital Ground	DGND	**19**	**20**	DGND	Digital Ground

3.7.3 I/O Control Connector(mirror) - J3

+5V power supply	VDD	**1**	**2**	VDD	+5V power supply
SPI Data In	SI	**3**	**4**	SO	SPI Data Out
SPI Clock	SCLK	**5**	**6**	CSB3	Chip select #3
Chip select #2	CSB2	**7**	**8**	FSPD1	Control signal #1
Chip select #1	CSB1	**9**	**10**	FSPD0	Control signal #0
Reset Output Devices	RESOUT	**11**	**12**	CSB0	Chip select #0
Reset Input Devices	RESIN	**13**	**14**	INTB	Interrupt signal
Microcontroller Reset	/HRESET	**15**	**16**	CSB4	Chip select #4
+24V module power supply	VPWR	**17**	**18**	VPWR	+24V module power supply
Digital Ground	DGND	**19**	**20**	DGND	Digital Ground

3.7.4 CAN Connector - J4

CAN bus drive high line	CANH	**1**	**2**	CANL	CAN bus drive low line
Digital Ground	DGND	**3**	**4**	DGND	Digital Ground
CAN Receive Data	CANRXD	**5**	**6**	CANTXD	CAN Transmit Data
CAN wake up output	WAKEUP	**7**	**8**	N.C.	Not Connected

3.8 I/O Board Connectors

This section provides information of the I/O Board connectors pins assignment and meaning.

3.8.1 I/O Control Connector - J1

+5V power supply	VDD	**1**	**2**	VDD	+5V power supply
SPI Data In	SI	**3**	**4**	SO	SPI Data Out
SPI Clock	SCLK	**5**	**6**	CSB3	Chip select #3
Chip select #2	CSB2	**7**	**8**	FSPD1	Control signal #1
Chip select #1	CSB1	**9**	**10**	FSPD0	Control signal #0
Reset Output Devices	RESOUT	**11**	**12**	CSB0	Chip select #0
Reset Input Devices	RESIN	**13**	**14**	INTB	Interrupt signal
Not Connected	NC	**15**	**16**	NC	Not Connected
+24V module power supply	VPWR	**17**	**18**	VPWR	+24V module power supply
Digital Ground	DGND	**19**	**20**	DGND	Digital Ground

3.8.2 Digital Outputs Connector - J8

1	PWR_24V	
2	O0	Digital Otput #0
3	O1	Digital Otput #1
4	O2	Digital Otput #2
5	O3	Digital Otput #3
6	O4	Digital Otput #4
7	O5	Digital Otput #5
8	O6	Digital Otput #6
9	O7	Digital Otput #7
10	O8	Digital Otput #8
11	O9	Digital Otput #9
12	O10	Digital Otput #10
13	O11	Digital Otput #11
14	O12	Digital Otput #12
15	O13	Digital Otput #13
16	O14	Digital Otput #14
17	O15	Digital Otput #15
18	DGND	Digital Ground

3.8.3 Digital Inputs Connector - J9

1	PWR_24V	
2	I0	Digital Input #0
3	I1	Digital Input #1
4	I2	Digital Input #2
5	I3	Digital Input #3
6	I4	Digital Input #4
7	I5	Digital Input #5
8	I6	Digital Input #6
9	I7	Digital Input #7
10	I8	Digital Input #8
11	I9	Digital Input #9
12	I10	Digital Input #10
13	I11	Digital Input #11
14	I12	Digital Input #12
15	I13	Digital Input #13
16	I14	Digital Input #14
17	I15	Digital Input #15
18	DGND	Digital Ground

3.9 Memory Map

Table 3-2 shows the device memory map of the MC9S12DP256 after reset. Note that after reset the bottom 1k of the EEPROM ($0000 - $03FF) is hidden by the register space.

Table 3-2. MC9S12DP256 Memory Map

From	To	Size	Content
0x0000	0x03FF	1k	REGISTERS
0x0000	0x0FFF	4k	EEPROM
0x1000	0x3FFF	12k	RAM
0x4000	0x7FFF	16k	Fixed Flash
0x8000	0xBFFF	16k	Paged Flash
0xC000	0xFEFF	16k-256b	Fixed Flash
0xFF00	0xFFFF	256bytes	VECTORS

The internal register block, RAM, and EEPROM have default locations within the 64K byte standard address space, but may be reassigned to other locations during program execution by setting bits in the mapping registers. For a detailed description of the MC9S12DP256 memory map, refer to the MC9S12DP256 Device User Guide, Motorola document order number 9S12DP256BDGV2/D.

Section 4. Software Module Descriptions

4.1 Contents

4.2 Introduction

This section of the reference design provides a complete documentation of the Industrial CAN I/O Module Reference Design software.

4.2.1 Software Basics

All embedded software of this project was written using the CodeWarrior for MOTOROLA 8- & 16-Bit MCU version 1.0 by Metrowerks Corporation (http://www.metrowerks.com).

The msCAN Driver Software version 1.0 by Metrowerks was used for the development, as it is msCAN periphery low level driver. Although this tool made the development faster and the final source code more readable, it is assumed that the reader has at least some experience with Controller Area Network (CAN) connectivity. For CAN 2.0 specification see CAN in Automation (CiA) at http://www.can-cia.de/can/.

Software content of the project can be divided into two basic groups. In the first group, are all the initialization routines necessary to configure both the MCU peripherals and all the sub-modules of the system, while second part consists of the routines for the Industrial CAN I/O Module Reference Design demo application. Therefore, separate descriptions will be given for the initialization routines and for the demo application. The demo application is enabled when the following symbolic constant is defined in **CAN_slave.h** file.

```
#define BLACK_BOX        /* if defined, the Black Box demo
application is ON */
```

As mentioned in previous chapters, the Industrial CAN I/O Module Reference Design is a modular system logically divided into the following three basic boards:

- Base Board
- Power Supply
- I/O Board

The Base Board is logically divided into the following four basic blocks:

- Micro-controller
- CAN Interface
- Analog Inputs
- RS232_485 Interface

4.2.2 Initialization Routines Basics

Here is a list of the routines for the first group, responsible for initialization and configuration of the Industrial CAN I/O Module

Reference Design. Details about each item on the list will be given in the **Software Implementation** chapters.

- MOTOROLA Scalable CAN (msCAN) periphery module initialization (including module addressing, baud-rate selection, CAN identifiers settings and msCAN message objects definition)
- Configuration of the CAN Interface block of the Base Board, via the SPI channel
- Analog to Digital converter (ATD) periphery module initialization (including ATD conversion accuracy)
- Configuration of the Analog Inputs block of the Base Board (voltage / current loop mode setting and voltage range selection in voltage mode)
- Serial Peripheral Interface (SPI) periphery module initialization for communication with devices of I/O Board
- Configuration of the I/O Board (Switch Monitor Interface MC33884 as device for digital inputs, and Octal Serial Switch MC33298 as digital outputs device) via the SPI channel
- Configuration of the PC33394 Multi-output Power Supply with integrated high speed CAN transceiver via the SPI channel
- Serial Communication Interface (SCI) periphery module initialization for RS232_485 Interface block of the Base Board
- Real Time Interrupt (RTI) module initialization for demo application

4.2.3 Demo Application Basics

Detailed introduction to the application is given in **Section 2. Quick Start**. That chapter describes software installation, demo setup & configuration, and even the PC Master software shared variables. Hence **Software Module Descriptions** chapter will mainly provide information of the software implementation about the Industrial CAN I/O Module Reference Design.

NOTE: *Please note that for the Industrial CAN I/O Module Reference Design demo application, both the Industrial CAN I/O Module Reference Design as well as the PC_CAN Interface have to be used. However, a standalone Industrial CAN I/O Module Reference Design provides an Input/ Output module for industrial automation purpose. Besides this, the Industrial CAN I/O Module Reference Design can be used as a hardware platform for any other user software development, such as high level communication protocol implementation and so on.*

The aim of the demo application is to show the main features of the Industrial CAN I/O Module Reference Design (16 digital and 8 analog inputs, 16 digital outputs), together with the utilization of the CAN connectivity. A simplified description of the Industrial CAN I/O Module Reference Design part of the application is that the module receives the configuration messages from the Superior device (PC_CAN Interface), and returns information about the status of its own inputs / outputs. A list of all events of the module can be seen in the following **Table 4-1**.

Table 4-1. List of application events

Event description	Initiator	Recipient
Change of Analog Inputs block parameters (range, accuracy) of Industrial CAN I/O Module Reference Design	Superior device	Industrial CAN I/O Module Reference Design
Status of Analog Inputs block values of Industrial CAN I/O Module Reference Design	Industrial CAN I/O Module Reference Design	Superior device
Set new Digital Outputs values of Industrial CAN I/O Module Reference Design	Superior device	Industrial CAN I/O Module Reference Design
Status of all Digital values of Industrial CAN I/O Module Reference Design	Industrial CAN I/O Module Reference Design	Superior device

Table 4-1. List of application events

Event description	Initiator	Recipient
Change of state of Digital Inputs values of Industrial CAN I/O Module Reference Design	Industrial CAN I/O Module Reference Design	Superior device

4.3 Project Introduction

This section gives an introduction and description of the software implementation of the Industrial CAN I/O Module Reference Design project.

4.3.1 List of the Project Files

The project was written using the Metrowerks CodeWarrior for MOTOROLA 8- & 16-Bit MCU version 1.0. In this chapter, a list of all source code files of the CodeWarrior project can be found. It will be divided into three parts:

- Project source codes
- MC9S12DP256 periphery structure
- msCAN Driver Software routines

4.3.1.1 Project Source Codes

- **slave CAN.mcp** is a Metrowerks CodeWarrior project file
- **CAN_slave.c** is a central file of the project, containing the complete initialization, global variables declaration and the *main()* routine
- **CAN_slave.h** is the header file of the **CAN_slave.c**; it contains the whole set of application-related symbolic constants, as well as the structure definitions
- **spi.c** and **spi.h** consist of Serial Peripheral Interface (SPI) based routines used for periphery module initialization and communication with MC33884, MC33298 and PC33394 devices

- **rti.c** and **rti.h** contain all Real Time Interrupt (RTI) related routines, specifically the periphery initialization and interrupt service routine (ISR)
- **atd.c** and **atd.h** include all Analog to Digital (ATD) based periphery routines used in project
- **sci.c** and **sci.h** files contain Serial Communications Interface (SCI) initialization routine
- **io.c** incorporates initialization of all general purpose inputs and outputs (GPIO) of the project; these are found in the Port Integration Module (PIM) as well as in the Bus Control and Input / Output
- **io.h** is a header file of **io.c** containing all pin mappings and GPIO related function style macros of the project
- **s12_regs.c** and **s12_regs.h** are files for periphery module allocation (see 4.3.1.2 MC9S12DP256 Periphery Structure) within MCU memory
- **MC9S12DP256_FLAT.prm** and **MC9S12DP256_RAM.prm** are parameter files of the device for RAM and FLASH configuration

4.3.1.2 MC9S12DP256 Periphery Structure

- **s12_atd.h** is a header file for Analog to Digital (ATD) register block
- **s12_bdlc.h** is a header file for J1850 Byte Data Link Controller (BDLC)
- **s12_common.h** is a header file for HCS12 common definitions
- **s12_crg.h** is a header file for HCS12 Clocks and Reset Generator (CRG) block
- **s12_eeprom.h** is a header file containing EEPROM control registers block definitions of HCS12
- **s12_flash.h** is a header file for HCS12 Flash control registers block
- **s12_iic.h** is header file for HCS12 Inter-IC Bus (IIC) register block

- **s12_mscan.h** is a header file for HCS12 Motorola Scalable (msCAN) register block
- **s12_page.h** is a header file for HCS12 Page (MEBI) register block
- **s12_pim.h** is a header for HCS12 Port Integration Module (PIM) block
- **s12_pwm.h** is a header file for HCS12 PWM register block
- **s12_register.h** is a header file for HCS12 register block
- **s12_sci.h** is a header file containing HCS12 Serial Communications Interface (SCI) register block definitions
- **s12_spi.h** is a header file for HCS12 Serial Peripheral Interface (SPI) register block definition
- **s12_template.h** is a template file of periphery register block definition
- **s12_timer.h** is a header file for HCS12 Timer block

NOTE: *Note that although not all of the MC9S12DP256 periphery modules are used within the project, all of them are included in the project. Thus they are available to a developer for immediate use.*

4.3.1.3 msCAN Driver Software Routines

As already mentioned, msCAN Driver Software is used in the project, and thus msCAN low level initialization is eliminated from the project. The driver itself consists of a couple of source code files (*.c and *.h) and one object file called **msCANs12drv.o,** where a key part of the driver implementation is carried out. Chapter **4.4.2** covers all the msCAN related topics to reference design.

4.3.1.4 MCU Peripherals Utilized

This section briefly describes all MCU peripheral components used in the project. It gives an overall summary picture of the necessary MCU resources.

Usage of the Analog to Digital (ATD) converter module is given in the table **Table 4-2**. Only ATD0 periphery is used in the reference design, and the module is set to scan across all eight available channels.

Table 4-2. ATD0 module

MCU pin	Symbolic name of the signal	Purpose	ISR function
AN07	AO0	data conversion of AO0 signal	-
AN06	AO1	data conversion of AO1 signal	-
AN05	AO2	data conversion of AO2 signal	-
AN04	AO3	data conversion of AO3 signal	-
AN03	AO4	data conversion of AO4 signal	-
AN02	AO5	data conversion of AO5 signal	-
AN01	AO6	data conversion of AO6 signal	-
AN00	AO7	data conversion of AO7 signal	-

NOTE: *ATD1 periphery module is not used in the reference design.*

A brief description of the SPI modules usage is given in the following **Table 4-3**.

Table 4-3. SPI modules usage

SPI	Purpose	ISR function
SPI0	communication with MC33884, MC33298 and PC33394 devices	-
SPI1	not used	n/a
SPI2	not used	n/a

A description of the SCI modules usage is given in the following table.

Table 4-4. SCI modules usage

SCI	Purpose	ISR function
SCI0	RS232_485 Interface communication	-
SCI1	not used	n/a

NOTE: *SCI0 periphery module is only initialized and not used anywhere within the project.*

A description of the msCAN modules usage is given in the **Table 4-5**.

Table 4-5. msCAN modules usage

msCAN	Purpose	ISR function
msCAN0	demo application CAN connectivity	authority of msCAN Driver Software
msCAN1	not used	n/a
msCAN2	not used	n/a
msCAN3	not used	n/a
msCAN4	not used	n/a

4.3.2 Utilized Interrupts

All interrupts used within the Industrial CAN I/O Module Reference Design project are briefly detailed in **Table 4-6**.

Table 4-6. Interrupts

Symbolic name of periphery	ISR function	Type of the interrupt	Note
RTI	`rtiISR()`	real time interrupt	n/a

Table 4-6. Interrupts

Symbolic name of periphery	ISR function	Type of the interrupt	Note
msCAN0 Tx	`CAN0_TransmitISR()`	msCAN transmission	authority of msCAN Driver Software
msCAN0 Rx	`CAN0_ReceiveISR()`	msCAN reception	authority of msCAN Driver Software
msCAN0 wake-up	`CAN0_WakeupISR()`	msCAN wake-up	authority of msCAN Driver Software

4.3.3 Project Variables

In this section a brief description of the main project variable is given. This variable is called *node* and is declared as follows:

```
volatile sNode node;           /* complete Node information */
```

This single structure variable contains complete information of the device, such as node identification (*node.nodeID*), values of digital inputs and digital outputs (*node.digitIn*, *node.digitOut*), analog input values (*node.analog[8]*) and configuration of the analog inputs (*node.analogConf[8]*). *sNode* structure is defined as follows:

```
typedef struct                  /* structure of analog[8].struc variable word */
{
    tU16 value : 10;    /* analog value of ADC */
    tU16 dumb  : 5;     /* reserved for future */
    tU16 mode  : 1;     /* mode configuration of ADC module */
        /* mode = 0 ... normal voltage measurement according to "range" value */
        /* mode = 1 ... current loop measurement */
} sAnalog;

typedef struct                  /* structure of analogConf byte variable */
{
    tU08 range : 2;     /* range configuration of ADC module */
    tU08 dumb  : 5;     /* reserved for future */
    tU08 accuracy : 1;  /* ADC module accuracy */
        /* accuracy = 0 ... 8 bit accuracy */
        /* accuracy = 1 ... 10 bit accuracy */
} sAnalogConf;

typedef union                  /* union for analog word variable  */
{
  tU16 word;             /* access whole word */
  struct                 /* access byte at a time */
  {
    tU08 msb;
    tU08 lsb;
  } byte;
  sAnalog   struc;       /* access as declared in sAnalog structure */
} uAnalog;

typedef union                  /* union for digital variable word */
{
  tU16 word;             /* access whole word */
  struct                 /* access byte at a time */
  {
    tU08 msb;
    tU08 lsb;
  } byte;
} uDigital;

typedef struct                  /* structure of the node information */
{
    tU08        nodeID;         /* node identification */
    uDigital    digitIn;        /* digital input values */
    uDigital    digitOut;       /* digital output values */
    uAnalog     analog[8];      /* union of analog value */
    sAnalogConf analogConf[8];  /* analog configuration structure */
} sNode;
```

NOTE: *For a graphical representation of the key components of sNode structure type see* ***4.4.6.2 Message Types Details****.*

There are also a couple of symbolic constants (defined in **CAN_slave.h**), which control the behaviour and configuration of the application.

- The demo application is enabled when the *BLACK_BOX* symbolic constant is defined in file.
- When a 16 MHz crystal is utilized in the design, the *OSC_16MHZ* symbolic constant has to be defined; for 4 MHz, the *OSC_4MHZ* definition should be used.

- When *FAST_CAN_ENABLE* symbolic constant is not defined, the MC33388D Low speed CAN physical line driver (125kbps) is used, instead of the PC33394 Multi-output Power Supply with integrated high speed CAN transceiver.

```
/******************************************************************************/
/*              A P P L I C A T I O N   D E F I N E S                     */
/******************************************************************************/
/* public defines for user's reconfiguration */
#define OSC_16MHZ       /* running on 16Mhz crystal */
//#define OSC_4MHZ         /* running on 4MHz crystal */

#define BLACK_BOX       /* if defined, the Black Box demo application is ON */

#define FAST_CAN_ENABLE /* if defined, MC33394 Power Oak is used instead of
   MC33388D low speed CAN physical line driver */
```

NOTE: *Note that the variable (higher) CAN baudrate settings (via dedicated DIP switch) can be used only when both the PC33394 device is connected and OSC_16MHZ is defined (16 MHz crystal connected), otherwise the CAN baudrate is fixed at 125kbps */*

4.3.4 Memory Usage

The following table shows the Industrial CAN I/O Module Reference Design software memory usage:

Table 4-7. Memory usage

Type of memory	Total size (B)	Used memory (B)
program flash	40000_h	$10A1_h$
data	3000_h	$22F_h$

4.4 Software Implementation

In this section a complete description of the key software modules for the reference design is given.

4.4.1 SPI Communication

The configuration of the Multi-output Power Supply with integrated high speed CAN transceiver (PC33394), and configuration and control of the Switch Monitor Interface MC33884 (as digital inputs) and Octal Serial Switch MC33298 (for digital outputs) devices, are done through the SPI channel. It was therefore necessary to find out an SPI channel configuration that is as efficient as possible and valid for all of the connected devices. But because of the fact that each device was originally developed for different customers there are a couple of dissimilar SPI format related parameters. A description of the SPI module initialization is presented in the next chapter.

4.4.1.1 SPI Periphery Module Initialization

The initialization is implemented in *spi0Init()* routine (**spi.c**) which is a part of the main *init()* function of **CAN_slave.c** file.

There are a couple of key settings of the SPI format. At first, it is important to set the *Master mode of the SPI device* to equal 1, because only a master SPI device can initiate transmission with peripherals. The SPI baud rate setting (value of the SPI Serial clock, called SCLK) has to be set to a value suitable for each of the three connected devices; SCLK frequency equal to 4 MHz was chosen.

For the SPI communication, the pooling approach was chosen so there is no SPI interrupt enabled.

Here is the complete listing of the *spi0Init().*

NOTE: *Some parts of the listing are discussed further within this chapter.*

```
/*******************************************************************************
*
* Module: void spi0Init(void)
*
* Description: The SPI channel is used for communication with MC33884, MC33298
*       and Power Oak MC33394. (Power Oak is used when FAST_CAN_ENABLE symbolic
*       constant is defined, otherwise the MC33388D device is used instead.)
*       This routine configures the SPI communication parameters.
*       Finally it configures:
*           - Switch Monitor Interface MC33884 into operation.
*           - Power Oak MC33394 device (if enabled)
```

```
*
* Special Issues: GPIO initialization has to be done before
*
*******************************************************************************/
void spi0Init(void)
{
    tU08 tmp;

    spi0.spicr1.bit.lsbf  = 0;      /* lsb first enable bit */
                                    /* msb bit is transferred first */
    spi0.spicr1.bit.ssoe  = 0;      /* slave select output enable */
                                    /* slave select output is not enabled */
    spi0.spicr1.bit.cpha  = 1;      /* SPI clock phase bit */
                                    /* first SCLK edge issued at the beginning
                                       of the 8-cycle transfer operation */
    spi0.spicr1.bit.cpol  = 0;      /* clock polarity bit */
                                    /* serial clock (SCK) active in high, SCK idles
low */
    spi0.spicr1.bit.mstr  = 1;      /* master/slave mode select bit */
                                    /* Master mode selected */
    spi0.spicr1.bit.sptie = 0;      /* transmit interrupt enable bit */
                                    /* transmit interrupt disabled, SPI
                                       communication done in pooling  style */
    spi0.spicr1.bit.spe   = 1;      /* spi enable bit */
                                    /* enable SPI, SPI port pins are dedicated to SPI
module */
    spi0.spicr1.bit.spie  = 0;      /* spi interrupt enable bit */
                                    /* SPI interrupt disabled */

    spi0.spicr2.bit.spc0    = 0;    /* serial pin control 0 bit */
                                    /* no bidirectional pin configuration of the SPI
*/
    spi0.spicr2.bit.spiswai = 0;    /* SPI stop in wait mode bit */
                                    /* SCLK operates normally in wait mode */
    spi0.spicr2.bit.bidiroe = 0;    /* bi-directional mode output enable bit */
                                   /* output buffer disable in bidirectional mode */
    spi0.spicr2.bit.modfen  = 0;    /* mode fault enable bit */
                                    /* disable the MODF error */

/* in order to run the SCLK on 4MHz while 16MHz crystal is connected
      (thus 8 MHz Module CLK), SPI module clock divisor has to be 2 */
    /* divider is set to 2, so SCLK is 4MHz for 8MHz Module Clk */
    spi0.spibr.bit.spr  = 0;        /* baud rate selection */
    spi0.spibr.bit.sspr = 0;        /* baud rate pre-selection */

    /* initialize both MC33884 chips for the first time */
    tmp = spi0TxWord(CSB2, TRISTATECMD);  /* first, tri-state command */
    tmp = spi0TxWord(CSB2, METALLICCMD);  /* then, metallic command */
    tmp = spi0TxWord(CSB2, RUNCMD);       /* finally, run command */

    tmp = spi0TxWord(CSB3, TRISTATECMD);
    tmp = spi0TxWord(CSB3, METALLICCMD);
    tmp = spi0TxWord(CSB3, RUNCMD);

#ifdef FAST_CAN_ENABLE
    /* initialize Power Oak MC33394 device for the first time */
    tmp = spi0TxWord(CSB4, POWEROAKCMD);  /* configure Power Oak */
#endif
}
```

As already mentioned there are couple of different SPI channel settings applicable for devices. First of them is the SPI format; while for MC33298 (Octal Serial Switch) and MC33884 (Switch Monitor Interface) the most significant bit is transferred first, for the PC33394 the least significant bit is transferred first. The second divergence is in the *SPI Clock phase shift bit* settings; for PC33394 and MC33884 devices this value has to be equal to 0 (the first SCLK edge is issued one-half cycle into the 8-cycle transfer operation), while MC33298 device needs that value to be 1 (the first SCLK edge is issued at the beginning of the 8-cycle transfer operation). And the last difference is the fact that Octal Serial Switch MC33298 device uses 8-bit long SPI format of communication, while the remaining two devices require a 16-bit long format.

These function style macros are used for proper SPI channel initialization before each communication with a particular device.

```
/* re-initialization of the SPI communication format for different devices:
   MC33298 (digital outputs):
        bit LSBF = 0    (lsb first enable)
        bit CPHA = 1    (clock phase bit, first SCLK edge at begin)
   MC33884 (digital inputs):
        bit LSBF = 0    (lsb first enable)
        bit CPHA = 0    (clock phase bit, first SCLK edge at begin)
   MC33394 (Power Oak)
        bit LSBF = 1    (lsb first enable)
        bit CPHA = 0    (clock phase bit, first SCLK edge at begin) */

#define setSPIForInputs()   spi0.spicr1.bit.cpha = 0;   \
                            spi0.spicr1.bit.lsbf = 0
#define setSPIForOutputs()  spi0.spicr1.bit.cpha = 1;   \
                            spi0.spicr1.bit.lsbf = 0
#define setSPIForPowerOak() spi0.spicr1.bit.cpha = 0;   \
                            spi0.spicr1.bit.lsbf = 1
```

NOTE: *Note that there is another difference in format of the SPI communication with devices. While for MC33884 and MC33298 the chip select signal is active in low, for PC33394 it is active in high.*

4.4.1.2 SPI Communication Routines

For the SPI communication, these two functions are implemented:

- *tU08 spi0TxByte (tU08 chipSelect, tU08 byte)* is used for the 8-bit long SPI communication with MC33298
- *tU16 spi0TxWord (tU08 chipSelect, tU16 cmd)* is used for the 16-bit long SPI communication suitable for MC33884 and PC33394 devices

```
/*******************************************************************************
*
* Module: tU08 spi0TxByte (tU08 chipSelect, tU08 byte)
*
* Description: This is the SPI communication function. It transmit one byte via
*       SPI and pass the received one as an argument.
*       This routine is used for the MC33298 device communication
*
* Returns: tmp as the SPI received byte
*
* Global Data: None
*
* Arguments:
*      chipSelect in order to choose the desired device. Possible values are CSB0 and
CSB1
*       byte as the value to be transmit
*
* Range Issues: possible values for chipSelect are CSB0 and CSB1 only
*
* Special Issues: Note that SPI format is different for each device. Proper
*       setting is done by calling proper function style macro (see spi.h for more)
*
*******************************************************************************/

/*******************************************************************************
*
* Module: tU16 spi0TxWord (tU08 chipSelect, tU16 cmd)
*
* Description: This is the SPI communication function. It transmit one word via
*       SPI and pass the received one as an argument.
*       This routine is used for the MC33884 and MC33394 "Power Oak" device
communication.
*      Note that Power Oak device has CSB active in high while MC33884 device in low.
*
* Returns: tmp as the SPI received word
*
* Global Data: None
*
* Arguments:
```

```
*       chipSelect in order to choose the desired device. Possible values
*           are CSB2 and CSB3 only (MC33884) and CSB4 for MC33394 Power Oak
*       cmd as the value to be transmit
*
* Range Issues: possible values for chipSelect are CSB2 and CSB3 only (MC33884)
*       and CSB4 for MC33394 Power Oak
*
* Special Issues: Note that SPI format is different for each device. Proper
*       setting is done by calling proper function style macro (see spi.h for more)
*
******************************************************************************/
```

4.4.1.3 MC33884, MC33298 and PC33394 Communication

For the Octal Serial Switch MC33298 device there is no initialization required. For the control of the module this command is used to load new values to the digital outputs; *CSB0* and *CSB1* (see **io.h**) chip select signals are linked with this type of device.

```
tmp = spi0TxByte(CSB0, node.digitOut.byte.lsb);
tmp = spi0TxByte(CSB1, node.digitOut.byte.msb);
```

For more information about this device, see MC33298 Octal Serial Switch **with serial peripheral interface I/O,** MOTOROLA **data sheet** MC33298**/D**.

For the initialization of both Switch Monitor Interface MC33884 devices the following technique is used. *CSB2* and *CSB3* (see **io.h**) chip select signals are linked with this type of device.

```
/* initialize both MC33884 chips for the first time */
    tmp = spi0TxWord(CSB2, TRISTATECMD);  /* first, tri-state command */
    tmp = spi0TxWord(CSB2, METALLICCMD);  /* then, metallic command */
    tmp = spi0TxWord(CSB2, RUNCMD);       /* finally, run command */

    tmp = spi0TxWord(CSB3, TRISTATECMD);
    tmp = spi0TxWord(CSB3, METALLICCMD);
    tmp = spi0TxWord(CSB3, RUNCMD);
```

Definition of symbolic constants of MC33298 commands is as follows:

```
/* defines for MC33884 device SPI communication */
#define TRISTATECMD 0x33FF
/* Tri-state command, enable SG1-SG4 and SP1-SP4 inputs */
#define METALLICCMD 0x5FFF
```

```
/* Metallic command, wetting current pulse enabled for SG1-SG4 and SP1-SP4 */

#ifdef BLACK_BOX     /* run command differs if "black box" is enabled or not */
    #define RUNCMD      0x140C
/* Run cmnd; normal mode set, lowest 2 prog. SPx switches set to ground, next 2 set
to battery */
#else
    #define RUNCMD      0x140F
/* Run command; normal mode set, 4 prog. SPx switches set to battery */
#endif
```

For the control of the module, this command is used to read new values of the digital inputs.

```
status = spi0TxWord(CSB2, RUNCMD);
currentDigitIn = (status) & 0xFF;
status = spi0TxWord(CSB3, RUNCMD);
currentDigitIn |= (status << 8) & 0xFF00;
```

For more information about this device see MC33884 Switch Monitor Interface**,** MOTOROLA **data sheet** MC33884**/D**.

When PC33394 device is enabled, its initialization is as follows, otherwise MC33388D Low speed CAN physical line driver is used. In that case no initialization is required.

```
#ifdef FAST_CAN_ENABLE
    /* initialize Power Oak MC33394 device for the first time */
    tmp = spi0TxWord(CSB4, POWEROAKCMD);  /* configure Power Oak */
#endif
```

For more information about this device see PC33394 Multi-output Power Supply with integrated high speed CAN transceiver**,** MOTOROLA **data sheet** PC33394**/D**.

4.4.2 msCAN Module

For the msCAN module, the msCAN Driver Software was successfully used to create more readable initialization and application routines while rapidly reducing total cycle time.

NOTE: *The msCAN Driver software may be ordered with the following part number: "MSCANDRV12".*

msCAN12 Low Level Drivers - Supports Motorola M68HC(S)12

For msCAN support queries, please email: CAN@metrowerks.com

4.4.2.1 msCAN Driver Initialization Introduction

The driver itself consists of several *.c and *.h files, and **msCANs12drv.o** object file, with the main implementation of the driver. For each key msCAN parameter of the periphery, there is a certain symbolic constant located in one of its header files. For more information see msCAN Driver Software 1.0, User Manual, Metrowerks.

In **msCAN0drv.h** the following settings related to the msCAN parameters can be found:

- Number of Message Buffers for msCAN module 0
- Clock prescaler for msCAN module 0
- msCAN module 0 bit timing
- Message Object Acceptance Filter size for msCAN module 0
- Message Object Acceptance Code for msCAN module 0
- Message Object Acceptance Filter Mask for msCAN module 0

In **msCAN0ID.h** the following settings related to the msCAN message objects (MO) identifiers can be found:

- Number of MO identifiers for msCAN module 0
- Message type (*STANDARD* or *EXTENDED*) declaration for each of MO identifiers
- all MO identifiers definition

Because of the fact that the application needs to change some of previously mentioned values during the program run, this small modification to the msCAN Driver Software code needs to be done:

- in **msCAN0ID.c** file, the *M_Identifier[NO_OF_ID_CAN0]* variable had to be placed in RAM to enable the program to change the message object identifier during the program run; see codelisting below

```
//const UINT32 M_Identifier_CAN0[NO_OF_ID_CAN0] =
/* in order to change value, this variable has to be placed in RAM */

UINT32 M_Identifier_CAN0[NO_OF_ID_CAN0] =
{
#if (NO_OF_ID_CAN0 > 0)
      MO0_IDR_CAN0
#endif

#if (NO_OF_ID_CAN0 > 1)
      ,MO1_IDR_CAN0
#endif
...
```

- similar trick is used in **msCANgvlite.c** file, with *CANBTR0_Def* variable, which has to be placed in RAM as well, to enable the program to change the CAN baudrate during the program run; see codelisting below

```
// const UINT8 CANBTR0_Def          = CANBTR0_CAN0;
/* in order to change value, this variable has to be placed in RAM */
UINT8 CANBTR0_Def                = CANBTR0_CAN0;
/* note that the default CANBTR0_CAN0 value has to be rewritten before CAN
   initialization routine call */
```

4.4.2.2 msCAN Driver Initialization

The CAN baudrate is preset in **msCAN0drv.h** to the value of 125 kbps. However, when *FAST_CAN_ENABLE* (PC33394 connected) and *OSC_16MHZ* (running on 16 Mhz) symbolic constants are defined, the user's CAN baudrate value is read and set according to the dedicated DIP switch; see codelisting below (part of *init()* routine of **CAN_slave.c** file)

```
#if defined FAST_CAN_ENABLE && defined OSC_16MHZ
    tmp = readDipBdr();     /* read DIP switch value for CAN baudrate */
    /* Possible values of readDipBdr(): value 0 - 125kbps
                                        value 1 - 250kbps
                                        value 2 - 500kbps
                                        value 3 - undefined, set 125kbps */

    /* modify CAN baudrate register according to the DIP Switch value */
    if (tmp == 0)
    CANBTR0_Def = (CANBTR0_Def & 0xB0) | (BAUDRATE_125 - 1);    /* 125kbps */
    else if (tmp == 1)
    CANBTR0_Def = (CANBTR0_Def & 0xB0) | (BAUDRATE_250 - 1);    /* 250kbps */
    else if (tmp == 2)
    CANBTR0_Def = (CANBTR0_Def & 0xB0) | (BAUDRATE_500 - 1);    /* 500kbps */
    else
    CANBTR0_Def = (CANBTR0_Def & 0xB0) | (BAUDRATE_125 - 1);    /* undefined */
                                              /* 125kbps as the default value */
#endif
/* when running on low speed CAN (MC33388D) baudrate is fixed at 125kbps */
/* when running on high speed CAN (Power Oak) but with 4MHz crystal, baudrate
   is fixed at 125kbps as well */
```

Message Object Acceptance Filter size for msCAN module 0 is set to be 16 bits long. However, Message Object Acceptance Filter Masks of msCAN module are all set to logical one, which means that all Message Object Acceptance Code is ignored, and the device receives ALL of the CAN messages on the network. This configuration can be used knowing that there is no other traffic on the network.

The application messages are all based on CAN 2.0 A 11-bit long identifiers, therefore, all used Message objects identifiers are set as *STANDARD* ones. However default values of the Message objects identifiers are ignored, because their values change as they are read from the dedicated DIP switch during the *init()* routine. This piece of code is responsible for the message identifier adjustments:

```
    node.nodeID = readDipNodeID();     /* read node identification number */
    shiftedNodeID = node.nodeID << 3;

/* Set CAN identifiers according to:    - key message identifiers
                                        - actual node ID address */
/* Note that this function was slightly modified from original msCAN Driver */
    M_Identifier_CAN0[0] = ((tU32)(CAN_KEYID_MSG_D | shiftedNodeID) << 21);
    M_Identifier_CAN0[1] = ((tU32)(CAN_KEYID_MSG_E | shiftedNodeID) << 21);
    M_Identifier_CAN0[2] = ((tU32)(CAN_KEYID_MSG_B | node.nodeID) << 21);
    M_Identifier_CAN0[3] = ((tU32)(CAN_KEYID_MSG_A1 | node.nodeID) << 21);
    M_Identifier_CAN0[4] = ((tU32)(CAN_KEYID_MSG_A2 | node.nodeID) << 21);
    M_Identifier_CAN0[5] = ((tU32)(CAN_KEYID_MSG_C | node.nodeID) << 21);
```

NOTE: *As a part of the message object identifier, the address of the device is used in the application. For more information about this topic see* ***Application*** *chapter, specially* ***Table 4-8. List of message types****.*

4.4.2.3 Initialization / Transmission / Reception Using msCAN Driver Software

For the **initialization of the** msCAN **module** the following routine of the msCAN Driver Software is used:

```
/* CAN init & configuration */
    tmp = CAN_Init(FAST, 0);
```

Next, it is necessary to configure employed message buffers (entities for handling CAN messages) using *CAN_ConfigMB()* function, the one message buffer for each message type (see **Table 4-8. List of message types**). This configuration consists of assigning the message type (message object identifier) and a direction of communication (reception or transmission) for each message buffer.

```
/* used CAN MO numbers plus letter names:
    Rx [letter identifier used in notation]:
    0 [D]  ... configure digital output msg - Msg Group 2 with msg group ID = 010
    1 [E]  ... analog configure msg - Msg Group 2 with msg group ID = 101

    Tx [letter identifier used in notation]:
    2 [B]  ... digital status msg - Msg Group 1 with msg group ID = 0100
    3 [A1] ... first analog status msg - Msg Group 1 with msg group ID = 1000
    4 [A2] ... scnd analog status msg - Msg Group 1 with msg group ID = 1001
    5 [C]  ... dig input change-of-state msg - Group 1 with msg group ID=0001 */

    tmp = CAN_ConfigMB(0, RXDF, 0, 0);  /* configure MO 0 to receive, ID = 0 */
        /* configure (set) digital outputs msg */
    tmp = CAN_ConfigMB(1, RXDF, 1, 0);  /* configure MO 1 to receive, ID = 1 */
        /* configure analog inputs msg */
    tmp = CAN_ConfigMB(2, TXDF, 2, 0);  /* configure MO 2 to transmit, ID = 2 */
        /* digital status msg */
    tmp = CAN_ConfigMB(3, TXDF, 3, 0);  /* configure MO 3 to transmit, ID = 3 */
        /* analog status msg, part 1 */
    tmp = CAN_ConfigMB(4, TXDF, 4, 0);  /* configure MO 4 to transmit, ID = 4 */
        /* analog status msg, part 2 */
    tmp = CAN_ConfigMB(5, TXDF, 5, 0);  /* configure MO 5 to transmit, ID = 5 */
        /* digital input change of state msg */
```

For the **CAN transmission** the following piece of code can be used. It prepares a two bytes long message in *sendData[]* buffer and sends it through message buffer number 5.

```
        tU08 sendData[9];                   /* pass data to CAN Tx routine */

        sendData[0] = 2;                    /* store desired data to sendData */
```

```
    sendData[1] = node.digitIn.byte.lsb;
    sendData[2] = node.digitIn.byte.msb;
    tmp = CAN_LoadMB(5, sendData, 0);   /* load buf */
    tmp = CAN_TransmitMB(5, 0);         /* send buf */
```

For the **CAN reception** the pooling technique is utilized in msCAN Driver Software. Therefore user code has to periodically check the status of each message buffer, by calling *CAN_CheckStatusMB()* function. When data arrives it can be read using *CAN_ReadDataMB()* function. The following demonstration piece of code checks the message buffers 0 and 1; when status *NEWDATA* is detected on them, the message buffer received a CAN message with a valid message object identifier.

```
tU08 bufSts[2];              /* buffer status of CAN reception */
tU08 bufData[9];             /* buffer of received CAN message */

tmp = CAN_CheckStatusMB(0, bufSts, 0); /* MB 0 - configure (set) digit out*/
if(bufSts[0] == NEWDATA)      /* new data in MB 0? */
{
    tmp = CAN_ReadDataMB(0, bufData, 0);
    node.digitOut.byte.lsb = bufData[1];    /* write new value */
    node.digitOut.byte.msb = bufData[2];
}
tmp = CAN_CheckStatusMB(1, bufSts, 0); /* Mb 1 - configure analog inputs */
if(bufSts[0] == NEWDATA)      /* new data in MB 1? */
{
    tmp = CAN_ReadDataMB(1, bufData, 0);
    ...
}
```

4.4.3 SCI Module Initialization

Although the SCI module is not utilized within the application, the project is already supplemented with SCI initialization function called *sci0Init(void)* (**sci.c** file), which enables both the transmitter and the receiver of the module and sets the SCI baudrate. When running with a 16 MHz crystal, the SCI baudrate is set to 38.400 bps, while for a 4 MHz crystal, the preset baudrate is equal to 9.600 bps.

4.4.4 ATD Module

In this chapter all the Analog to Digital (ATD) converter related routines are explained.

4.4.4.1 ATD Initialization

ATD is set to scan continuously across all eight available channels. Resolution of conversion is passed to the function through the function argument, 10 bit resolution is the default value of the project. Complete initialization routine is given here:

```
void atd0Init(tU08 accuracy)
{
/* ATD configuration */
    atd0.atdctl2.bit.adpu = 1;      /* ADPU turns on ATD */
    atd0.atdctl2.bit.affc = 1;
    /* AFFC turns on fast clear mode of sequence complete flag (SCF) */

    periphBitSet(S8C | S4C | S2C | S1C, &atd0.atdctl3.byte);
    /* number of conversion is 8 per sequence */
    atd0.atdctl3.bit.frz = 1;
    /* when BDM breakpoint come, current conversion is finished then freezed */
    atd0.atdctl4.byte = 0x3;        /* set total divisor to 8 on EVM,  module clock
is 8MHz,                                        so ATD conversion clock it 1MHz */

    if (accuracy == ACCURACY_8BIT)      /* if 8-bit resolution is selected */
        atd0.atdctl4.bit.res8 = 1;      /* RES8 bit is set */
    else atd0.atdctl4.bit.res8 = 0;     /* otherwise 10-bit resolution */

    periphBitSet(DJM | SCAN | MULT, &atd0.atdctl5.byte);
    /* DJM sets right justified mode, SCAN sets conversion to sequence
       continuously, MULT sets to sample accros many channels */
}
```

4.4.4.2 Configuration of Analog Input Block

As described in **3.4.1.3 The Analog Inputs** section, there are three signals (CSI, G and G/2) controlling each analog channel.

The *CSI* signal is a “wired” input which indicates if the analog channel is in *Voltage* (CSI is high) or *Current Sensing* (CSI is low) mode. This signal is read (and stored in proper place of the *node* structure) only once during the *init()* function execution, and is valid till next reset of the MCU.

```
    node.analog[0].struc.mode = pAIS0;  /* set mode according to jumper info */
    node.analog[1].struc.mode = pAIS1;  /* either normal voltage measurement */
    node.analog[2].struc.mode = pAIS2;  /* according to range struct value */
    node.analog[3].struc.mode = pAIS3;  /* or current loop measurement */
    node.analog[4].struc.mode = pAIS4;
    node.analog[5].struc.mode = pAIS5;
    node.analog[6].struc.mode = pAIS6;
    node.analog[7].struc.mode = pAIS7;
```

Next, it is necessary to set the input range of each analog channel. This piece of code is responsible for this operation.

```
    for (i = 0; i < 8; i++)
    {
        node.analogConf[i].accuracy = 1; /* 10 bit resolution default */
        node.analogConf[i].range = 1;    /* voltage range of 0V to 10V default */
    }
        /* set analog configuration values, bit by bit */
    pAIC0_0 = (node.analogConf[0].range & 1);
    pAIC0_1 = (node.analogConf[0].range & 2) >> 1;
    pAIC1_0 = (node.analogConf[1].range & 1);
    pAIC1_1 = (node.analogConf[1].range & 2) >> 1;
    pAIC2_0 = (node.analogConf[2].range & 1);
    pAIC2_1 = (node.analogConf[2].range & 2) >> 1;
    ...
```

NOTE: *Because of the fact that the demo project is utilizing only the voltage mode measurement, there is no implementation of the condition when current sensing mode is detected. In that case signals G and G/2 have to be set to logical one as mentioned in **Table 3-1. Input Range Control Signals**.*

4.4.4.3 ATD In Application

The ATD conversion result registers are read by the application, using *atd0Read()* function. It is regularly called, based on the RTI interrupt event. This routine reads ATD conversion results and saves them into *node.analog[]* variable. Note that Analog Input 0 signal is mapped to AN07 pin of MCU, Analog Input 1 signal to AN06, and so on.

```
void atd0Read(void)
{
    tU08 i;
    while(periphBitTest(SCF, &atd0.atdstat0.byte) == 0);  /* wait for flag */
    /* it is necessary to wait for a complete flag */

    for (i = 0; i < 8; i++)                              /* read values */
    {
        node.analog[i].struc.value = atd0.atddr[7 - i].word;
       /* write analog value, note that Analog Input 0 is mapped to AN07 pin
          of MCU, Analog Input 1 to AN06 and so on */
    }
}
```

4.4.5 RTI Module

In this chapter all Real time interrupt (RTI) related routines are explained.

4.4.5.1 RTI Initialization

Real time interrupt is set to interrupt the process 15.25 times per second. Complete initialization routine is given here:

```
void rtiInit(void)
{
#ifdef OSC_4MHZ
    crg.rtictl.byte = 0x73;     /* real time interrupt 15.25 times per second */
#endif
#ifdef OSC_16MHZ
    crg.rtictl.byte = 0x7F;     /* real time interrupt 15.25 times per second */
#endif

    crg.crgint.bit.rtie = 1;    /* real time interrupt enable */
}
```

4.4.5.2 RTI In Application

RTI interrupt service request routine is called *rtiISR()*. Although almost all timing events are dedicated for application related CAN transmission, there is also the ATD conversion function, *atd0Read()*, regularly called from RTI interrupt service routine. For more, see function header of the function below, and **Figure 4-7. rtiISR() function flowchart**.

```
/********************************************************************************
*
* Module: void rtiISR(void)
```

```
*
* Description: This routine is the interrupt service routine of the RTI module,
*       it interrupts the program 15.25 times per second.
*       It it used for CAN message sending (both analog and digital status msg)
*
*       When "black box" application is enabled, it is used for generation of
*       two CAN analog status messages 15.25 times per second + CAN digital
*       status message once per second approximately.
*       When "black box" application is disabled, it sends two CAN analog status
*       and one CAN digital messages once per second (distributed through the time)
*
* Returns: None
*
* Global Data:
*      BLACK_BOX is a symbolic constant, if defined the "black box" demo application
is enabled
*       node.analog
*       node.digitIn
*       node.digitOut
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues:
*       ADC conversion is done in pooling fashion.
*
********************************************************************************/
```

4.4.6 Application

This chapter summarizes the Industrial CAN I/O Module Reference Design part of the demo application for the reference design. Note that the complete application is running on the Industrial CAN I/O Module Reference Design, as well as on the PC_CAN Interface device linked with the CAN network.

4.4.6.1 Application Introduction

The application itself is separated into the following tasks, with more details given in the following chapters.

- CAN reception and proper handling of all types of configuration messages (message buffers 0 and 1)

- Detection of the change of state of the Digital Inputs (MC33884), and proper handling of that event via the dedicated CAN message (linked with message buffer number 5)
- Periodical signalling of the status of the Analog Inputs and all Digital variables via CAN messages (tied with message buffers number 2, 3 and 4); timing based on the Real Time interrupt of MCU

4.4.6.2 Message Types Details

A straightforward structure of the messages based on the CAN 2.0A 11-bit long identifiers was created, for all events listed in **Table 4-1. List of application events**. Every message type has a definition of its identifier as can be seen in **Table 4-8**. Note that the definition of the Group numbers and identifiers structure are chosen in correspondence with the DeviceNet specification, release 2.0, Open DeviceNet Vendor Association.

Table 4-8. List of message types

Message type	Message buffer number	Complete CAN identifier	Group number (of DeviceNet)	Message identifier (of DeviceNet)
Analog Inputs configuration	1	10aaaaaa101	2	101
Analog Inputs status - part 1	3	01000aaaaaa	1	1000
Analog Inputs status - part 2	4	01001aaaaaa	1	1001
Digital Output configuration	0	10aaaaaa010	2	010
Digital Inputs status	2	00100aaaaaa	1	0100
Digital Inputs status change of state	5	00001aaaaaa	1	0001

The sign “a” in CAN identifier (message object identifier) definition stands for one bit of Node Address (*NodeID*) as mentioned in **4.4.2.2 msCAN Driver Initialization**.

Table 4-9. Messages description

Message type	Message buffer number	Length of the message [in B]	Content
Analog Inputs configuration	1	8	see **Figure 4-1**
Analog Inputs status - part 1	3	8	see **Figure 4-2** for analog inputs 0 to 3
Analog Inputs status - part 2	4	8	see **Figure 4-2** for analog inputs 4 to 7
Digital Output configuration	0	2	see **Figure 4-3**
Digital status	2	2	first dig. inputs (**Figure 4-4**), then dig. outputs (**Figure 4-3**)
Digital Inputs change of state	5	4	see **Figure 4-4**

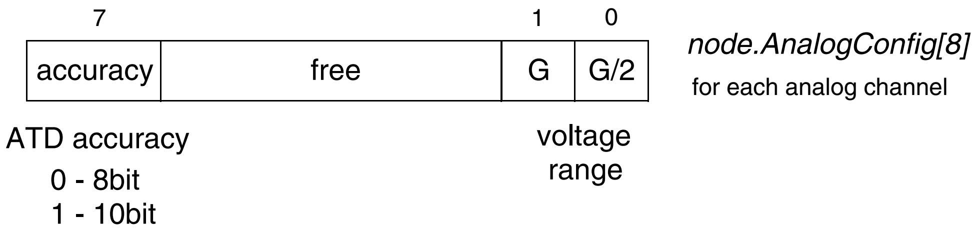


Figure 4-1. Analog Configuration byte composition

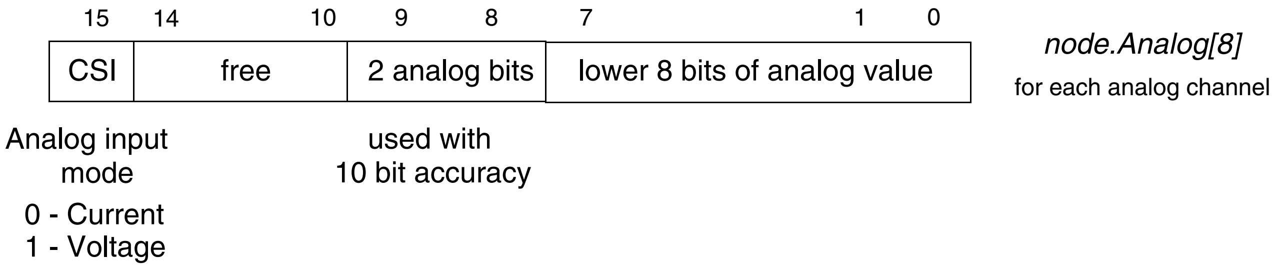


Figure 4-2. Analog Configuration word composition

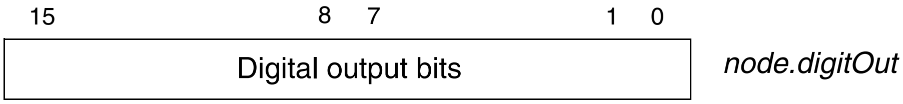


Figure 4-3. Digital Outputs word composition

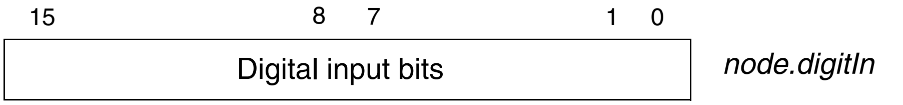


Figure 4-4. Digital Inputs word composition

4.4.6.3 Main routine details

Description of *main()* routine of the Industrial CAN I/O Module Reference Design reference design demo application can be found in this section, its codelisting is as follows.

```
void main(void)
{
   init(); /* Initialization of periphery modules & variables */

    while(1)
    {
        rxCANProcess();      /* CAN reception */
        digitInProcess();    /* Digital input testing */
    }
}
```

NOTE: *Note that there is also an RTI interrupt service routine involved within this project main loop.*

The CAN reception handling is carried out in *rxCANProcess()* routine; its flowchart is given in **Figure 4-5**. There are two types of CAN messages being received by Industrial CAN I/O Module Reference Design device in the application: analog channel configuration (message buffer 1), and configuration of digital outputs (message buffer number 0).

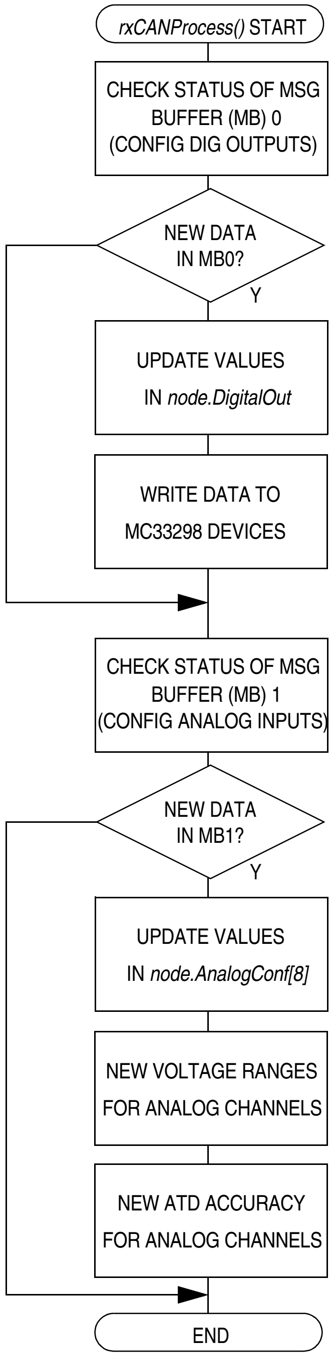

rxCANProcess() START
CHECK STATUS OF MSG BUFFER (MB) 0 (CONFIG DIG OUTPUTS)
NEW DATA IN MB0?
Y
UPDATE VALUES IN node.DigitalOut
WRITE DATA TO MC33298 DEVICES
CHECK STATUS OF MSG BUFFER (MB) 1 (CONFIG ANALOG INPUTS)
NEW DATA IN MB1?
Y
UPDATE VALUES IN node.AnalogConf[8]
NEW VOLTAGE RANGES FOR ANALOG CHANNELS
NEW ATD ACCURACY FOR ANALOG CHANNELS
END

Figure 4-5. *rxCANProcess()* function flowchart

Flowchart of the *digitInProcess()* routine is given in **Figure 4-6**. This function does the change of state detection of the digital inputs (two Switch Monitor Interface MC33884 devices) of Industrial CAN I/O Module Reference Design. When change is detected, a respective CAN message containing 16 bit long digital input information is sent to the PC_CAN Interface device.

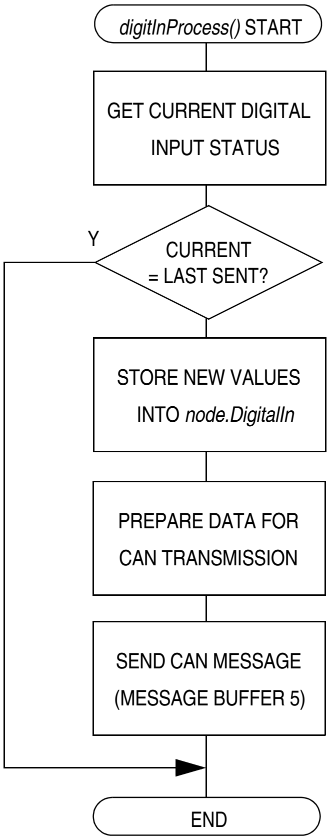

digitInProcess() START
GET CURRENT DIGITAL INPUT STATUS
CURRENT = LAST SENT?
Y
STORE NEW VALUES INTO *node.DigitalIn*
PREPARE DATA FOR CAN TRANSMISSION
SEND CAN MESSAGE (MESSAGE BUFFER 5)
END

Figure 4-6. *digitInProcess()* function flowchart

The RTI module interrupt service routine is called *rtiISR()*. It interrupts the program execution 15.25 times per second. This routine is used for CAN message sending for both analog and digital status messages. When "Black box" application is enabled, it is used for the generation of two CAN analog status messages, 15.25 times per second, and one CAN digital status message, approximately once per second (see **Figure 4-7** for routine flowchart when BLACK_BOX symbolic constant is defined).
When "Black box" application is disabled, it sends two CAN analog status and one CAN digital messages, once per second (evenly distributed through the time).

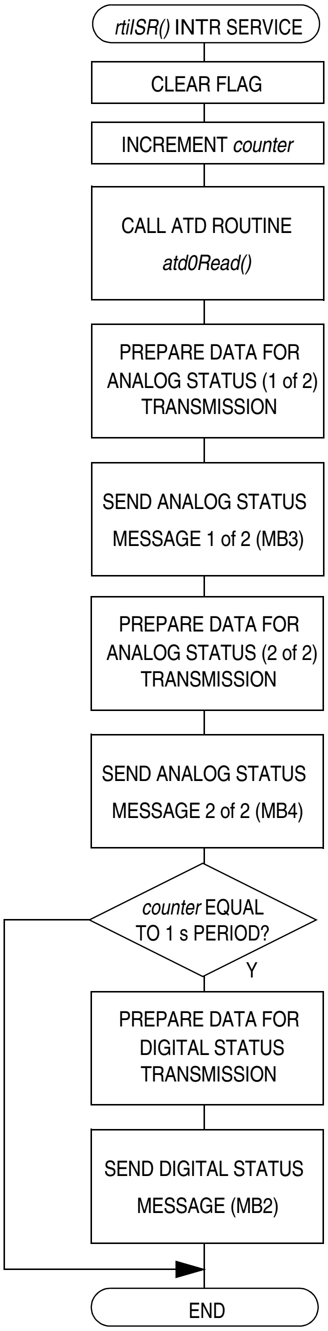


Figure 4-7. *rtiISR()* function flowchart

Appendix A. Source Code Files

A.1 Contents

A.2 CAN_slave.c

```
/*******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: CAN_slave.c
*
* Description: Code for the “Industrial CAN I/O module” project
*
* Modules Included:
*       init()
*       rxCANProces()
*       digitInProces()
*       main()
*
*******************************************************************************/
#include “s12_regs.h”       /* register definition */

#include “CAN_slave.h”      /* project main header file */
#include “atd.h”
#include “spi.h”
#include “sci.h”
#include “io.h”
#include “rti.h”

#include “msCANstd.h”       /* msCAN module */
#include “msCANdrv.h”


/******************************************************************************/
/*              P R O T O T Y P E S                                           */
/******************************************************************************/
```

```
void init(void);
void rxCANProcess(void);
void digitInProcess(void);


/******************************************************************************/
/*              G L O B A L   V A R I A B L E S                               */
/******************************************************************************/
volatile sNode node;            /* complete Node information */
tU08     tmp;                   /* temporary variable */

extern tU32 M_Identifier_CAN0[];  /* array of CAN identifiers of msCAN driver */
extern UINT8 CANBTR0_Def;         /* bitrate 0 CAN register value */
/* these two entities were originally constants of msCAN driver, but there were
   changed to be variables placed in RAM memory */


/*******************************************************************************
*
* Module: void init(void)
*
* Description: This routine initializes all used periphery modules: PIM & IO,
*       SPI, IRQ, RTI, SCI, msCAN.
*       Note that msCAN driver is used for msCAN periphery module handling.
*       Note that SCI module is not used, it is just initialized.
*
*       According to the DIP switch values, routine modifies:
*       - CAN baudrate (when Powek Oak connected as a high speed CAN transceiver)
*       - actual node ID (identification address)
*       It also configures the CAN message objects (MO) for 6 used message
*       buffers + configure CAN identifiers for those 6 msg buffers.
*       And finally set default values for all node variables.
*
* Returns: None
*
* Global Data:
*       FAST_CAN_ENABLE is a symbolic constant, if defined the Power Oak PC33394
*           (fast CAN physical layer driver) is used, otherwise MC33388D is used
*           as a slow CAN physical layer driver
*       OSC_16MHZ is a symbolic constant, when defined, 16MHz crystal is
*           connected to board
*       OSC_4MHZ is a symbolic constant, when defined 4MHz crystal is
*           connected to board
*       CANBTR0_Def
*       node.nodeID
*       node.analogConf[].accuracy
*       node.analogConf[].range
*       node.analog[]
*       node.digitOut
*
* Arguments: None
```

```
*
* Range Issues:
*       if defined FAST_CAN_ENABLED (Power Oak connected)
*           Crystal on 4MHz  - CAN baudrate is fixed at 125kbps
*           Crystal on 16MHz - CAN baudrate is variable (125, 250 and 500kbps)
*       if not defined FAST_CAN_ENABLED (MC33388D device connected)
*           Crystal on 4MHz  - CAN baudrate is fixed at 125kbps
*           Crystal on 16MHz - CAN baudrate is fixed at 125kbps
*
* Special Issues: None
*
*******************************************************************************/
void init(void)
{
    tU08 i;
    tU16 shiftedNodeID; /* temp shifted variable for Msg Group 2 */

/* I0 (GPIO and PIM) configuration */
    ioInit();

#if defined FAST_CAN_ENABLE && defined OSC_16MHZ
    tmp = readDipBdr();     /* read DIP switch value for CAN baudrate */
    /* Possible values of readDipBdr(): value 0 - 125kbps
                                        value 1 - 250kbps
                                        value 2 - 500kbps
                                        value 3 - undefined, set 125kbps */

    /* modify CAN baudrate register according to the DIP Switch value */
    if (tmp == 0)
    CANBTR0_Def = (CANBTR0_Def & 0xB0) | (BAUDRATE_125 - 1);    /* 125kbps */
    else if (tmp == 1)
    CANBTR0_Def = (CANBTR0_Def & 0xB0) | (BAUDRATE_250 - 1);    /* 250kbps */
    else if (tmp == 2)
    CANBTR0_Def = (CANBTR0_Def & 0xB0) | (BAUDRATE_500 - 1);    /* 500kbps */
    else
    CANBTR0_Def = (CANBTR0_Def & 0xB0) | (BAUDRATE_125 - 1);    /* undefined */
                                             /* 125kbps as the default value */
#endif
/* when running on low speed CAN (MC33388D) baudrate is fixed at 125kbps */
/* when running on high speed CAN (Power Oak) but with 4MHz crystal, baudrate
   is fixed at 125kbps as well */

    node.nodeID = readDipNodeID();     /* read node identification number */
    shiftedNodeID = node.nodeID << 3;

/* Set CAN identifiers according to: - key message identifiers
                                     - actual node ID address */
/* Note that this function was slightly modified from original msCAN Driver */
    M_Identifier_CAN0[0] = ((tU32)(CAN_KEYID_MSG_D | shiftedNodeID) << 21);
    M_Identifier_CAN0[1] = ((tU32)(CAN_KEYID_MSG_E | shiftedNodeID) << 21);
    M_Identifier_CAN0[2] = ((tU32)(CAN_KEYID_MSG_B | node.nodeID) << 21);
    M_Identifier_CAN0[3] = ((tU32)(CAN_KEYID_MSG_A1 | node.nodeID) << 21);
    M_Identifier_CAN0[4] = ((tU32)(CAN_KEYID_MSG_A2 | node.nodeID) << 21);
    M_Identifier_CAN0[5] = ((tU32)(CAN_KEYID_MSG_C | node.nodeID) << 21);

/* SCI0 configuration */
    sci0Init();
/* SPI0 configuration - being used for MC33298 and MC33884 */
    spi0Init();
/* ATD configuration, default setting is with 10-bit accuracy */
    atd0Init( ACCURACY_10BIT );
/* RTI - real time interrupt module */
    rtiInit();

/* IRQ setting */
    reg.intcr.bit.irqen = 0;   /* disable external IRQ */

/* CAN init & configuration */
    tmp = CAN_Init(FAST, 0);

   /* used CAN MO numbers plus letter names:
    Rx [letter identifier used in notation]:
    0 [D]  ... configure digital output msg - Msg Group 2 with msg group ID = 010
    1 [E]  ... analog configure msg - Msg Group 2 with msg group ID = 101

    Tx [letter identifier used in notation]:
    2 [B]  ... digital status msg - Msg Group 1 with msg group ID = 0100
    3 [A1] ... first analog status msg - Msg Group 1 with msg group ID = 1000
    4 [A2] ... scnd analog status msg - Msg Group 1 with msg group ID = 1001
    5 [C]  ... dig input change-of-state msg - Group 1 with msg group ID=0001 */

    tmp = CAN_ConfigMB(0, RXDF, 0, 0);  /* configure MO 0 to receive, ID = 0 */
        /* configure (set) digital outputs msg */
    tmp = CAN_ConfigMB(1, RXDF, 1, 0);  /* configure MO 1 to receive, ID = 1 */
        /* configure analog inputs msg */
    tmp = CAN_ConfigMB(2, TXDF, 2, 0);  /* configure MO 2 to transmit, ID = 2 */
        /* digital status msg */
    tmp = CAN_ConfigMB(3, TXDF, 3, 0);  /* configure MO 3 to transmit, ID = 3 */
        /* analog status msg, part 1 */
    tmp = CAN_ConfigMB(4, TXDF, 4, 0);  /* configure MO 4 to transmit, ID = 4 */
        /* analog status msg, part 2 */
    tmp = CAN_ConfigMB(5, TXDF, 5, 0);  /* configure MO 5 to transmit, ID = 5 */
        /* digital input change of state msg */

    archEnableInt();    /* enable interrupts */

/* initial (default) values of node variables */
    for (i = 0; i < 8; i++)
    {
        node.analogConf[i].accuracy = 1; /* 10 bit resolution default */
        node.analogConf[i].range = 1;    /* volt. range of 0V to 10V default */
    }
```

```
        /* set analog configuration values, bit by bit */
    pAIC0_0 = (node.analogConf[0].range & 1);
    pAIC0_1 = (node.analogConf[0].range & 2) >> 1;
    pAIC1_0 = (node.analogConf[1].range & 1);
    pAIC1_1 = (node.analogConf[1].range & 2) >> 1;
    pAIC2_0 = (node.analogConf[2].range & 1);
    pAIC2_1 = (node.analogConf[2].range & 2) >> 1;
    pAIC3_0 = (node.analogConf[3].range & 1);
    pAIC3_1 = (node.analogConf[3].range & 2) >> 1;
    pAIC4_0 = (node.analogConf[4].range & 1);
    pAIC4_1 = (node.analogConf[4].range & 2) >> 1;
    pAIC5_0 = (node.analogConf[5].range & 1);
    pAIC5_1 = (node.analogConf[5].range & 2) >> 1;
    pAIC6_0 = (node.analogConf[6].range & 1);
    pAIC6_1 = (node.analogConf[6].range & 2) >> 1;
    pAIC7_0 = (node.analogConf[7].range & 1);
    pAIC7_1 = (node.analogConf[7].range & 2) >> 1;

    node.analog[0].struc.mode = pAIS0;  /* set mode according to jumper info */
    node.analog[1].struc.mode = pAIS1;  /* either normal voltage measurement */
    node.analog[2].struc.mode = pAIS2;  /* according to range struct value */
    node.analog[3].struc.mode = pAIS3;  /* or current loop measurement */
    node.analog[4].struc.mode = pAIS4;
    node.analog[5].struc.mode = pAIS5;
    node.analog[6].struc.mode = pAIS6;
    node.analog[7].struc.mode = pAIS7;

    node.digitOut.word = 0;  /* default values: LEDs are switched on */
    tmp = spi0TxByte(CSB0, node.digitOut.byte.lsb);
    tmp = spi0TxByte(CSB1, node.digitOut.byte.msb);
}


/*******************************************************************************
*
* Module: void rxCANProcess(void)
*
* Description: This is the CAN reception routine, done in pooling style.
*       Message buffers 0 to 1 are configured as reception buffers.
*
*   buffer number [letter identifier used in notation]:
*   0 [D]  ... configure digital output msg - Msg Group 2 with msg group ID = 010
*   1 [E]  ... analog configure msg - Msg Group 2 with msg group ID = 101
*
* Returns: None
*
* Global Data:
*       node.digitOut
*       node.analogConf[].range
*       node.analogConf[].accuracy
*
```

```
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
//#pragma INLINE
void rxCANProcess(void)
{
    tU08 i;
    tU08 bufSts[2];             /* buffer status of CAN reception */
    tU08 bufData[9];            /* buffer of received CAN message */

    tmp = CAN_CheckStatusMB(0, bufSts, 0); /* MB 0 - configure (set) digit out*/
    if(bufSts[0] == NEWDATA)     /* new data in MB 0? */
    {
        tmp = CAN_ReadDataMB(0, bufData, 0);
        node.digitOut.byte.lsb = bufData[1];    /* write new value */
        node.digitOut.byte.msb = bufData[2];
        tmp = spi0TxByte(CSB0, node.digitOut.byte.lsb);
        tmp = spi0TxByte(CSB1, node.digitOut.byte.msb);
    }
    tmp = CAN_CheckStatusMB(1, bufSts, 0); /* MB 1 - configure analog inputs */
    if(bufSts[0] == NEWDATA)     /* new data in MB 1? */
    {
        tmp = CAN_ReadDataMB(1, bufData, 0);
        for (i = 0; i < 8; i++)                         /* write new values */
        {
            node.analogConf[i].range = (bufData[i + 1] & 0x3);
            node.analogConf[i].accuracy = bufData[i + 1] >> 7;
        }
        /* load analog configuration values, bit by bit */
        pAIC0_0 = (bufData[1] & 1);
        pAIC0_1 = (bufData[1] & 2) >> 1;
        pAIC1_0 = (bufData[2] & 1);
        pAIC1_1 = (bufData[2] & 2) >> 1;
        pAIC2_0 = (bufData[3] & 1);
        pAIC2_1 = (bufData[3] & 2) >> 1;
        pAIC3_0 = (bufData[4] & 1);
        pAIC3_1 = (bufData[4] & 2) >> 1;
        pAIC4_0 = (bufData[5] & 1);
        pAIC4_1 = (bufData[5] & 2) >> 1;
        pAIC5_0 = (bufData[6] & 1);
        pAIC5_1 = (bufData[6] & 2) >> 1;
        pAIC6_0 = (bufData[7] & 1);
        pAIC6_1 = (bufData[7] & 2) >> 1;
        pAIC7_0 = (bufData[8] & 1);
        pAIC7_1 = (bufData[8] & 2) >> 1;

        if (node.analogConf[0].accuracy == ACCURACY_8BIT)
```

```
            atd0Init(ACCURACY_8BIT);
        else atd0Init(ACCURACY_10BIT);
        /* if accuracy = 8BIT_ACCURACY (0x0), 8-bit resolution of ADC selected
           (RES8 bit set), otherwise 10BIT_ACCURACY (0xFF) 10-bit resolution */
    }
}


/********************************************************************************
*
* Module: void digitInProcess(void)
*
* Description: This routine read the digital input information from two MC33884
*       devices (Switch Monitor Interface).
*       When input values have changed, it send CAN message with actual digital
*       input information.
*
* Returns: None
*
* Global Data:
*       node.digitIn
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
********************************************************************************/
//#pragma INLINE
void digitInProcess(void)
{
    tU16 status;
    static tU16 lastDigitIn;    /* previous digital input word */
    tU16 currentDigitIn;        /* current digital input word */
    tU08 sendData[9];           /* pass data to CAN Tx routine */

    status = spi0TxWord(CSB2, RUNCMD);
    currentDigitIn = (status) & 0xFF;
    status = spi0TxWord(CSB3, RUNCMD);
    currentDigitIn |= (status << 8) & 0xFF00;
    if (currentDigitIn != lastDigitIn)      /* if changed */
    {
        node.digitIn.word = currentDigitIn; /* save new value */
        lastDigitIn = currentDigitIn;
        sendData[0] = 2;                    /* store desired data to sendData */
        sendData[1] = node.digitIn.byte.lsb;
        sendData[2] = node.digitIn.byte.msb;
        tmp = CAN_LoadMB(5, sendData, 0);   /* load buf */
        tmp = CAN_TransmitMB(5, 0);         /* send buf */
    }
```

```
}


/*******************************************************************************
*
* Module: void main(void)
*
* Description: This is the main routine of the “CAN I/O Industrial” module.
*       First, it calls the init() funtion
*       Than it polls the following:
*           - complete CAN reception routine
*           - Digital input change-of-state routine
*
* Returns: None
*
* Global Data: None
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
void main(void)
{
    init();         /* Initialization of periphery modules & variables */

    while(1)
    {
        rxCANProcess();      /* CAN reception */
        digitInProcess();    /* Digital input testing */
    }
}
```

A.3 CAN_slave.h

```
/*******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: CAN_slave.h
*
* Description: Header file for the project CAN_slave
*
* Modules Included: None
*
*******************************************************************************/
#ifndef _CAN_slave_H_
#define _CAN_slave_H_


/******************************************************************************/
/*              A P P L I C A T I O N   D E F I N E S                         */
/******************************************************************************/
/* public defines for user's redonfiguration */
#define OSC_16MHZ       /* running on 16Mhz crystal */
//#define OSC_4MHZ         /* running on 4MHz crystal */

#define BLACK_BOX       /* if defined, the Black Box demo application is ON */

#define FAST_CAN_ENABLE /* if defined, MC33394 Power Oak is used instead of
   MC33388D low speed CAN physical line driver */
/* note that the variable (higher) CAN baudrate settings (via DIP switch) can
   be used only when Power Oak is connected and OSC_16MHZ is defined (16MHz
   crystal connected), otherwise the CAN baudrate is fixed at 125kbps */


/******************************************************************************/
/*              A P P L I C A T I O N   D E F I N E S                         */
/******************************************************************************/
/* private ones */
/* CAN baudrates valid only for 16MHz crystal */
#define BAUDRATE_125    8       /* value for 125kbps CAN baudrate */
#define BAUDRATE_250    4       /* value for 250kbps CAN baudrate */
#define BAUDRATE_500    2       /* value for 500kbps CAN baudrate */

/* defines for MC33884 device SPI communication */
#define TRISTATECMD 0x33FF
/* Tri-state command, enable SG1-SG4 and SP1-SP4 inputs */
#define METALLICCMD 0x5FFF
```

```
/* Metallic command, wetting current pulse enabled for SG1-SG4 and SP1-SP4 */

#ifdef BLACK_BOX     /* run command differs if “black box” is enabled or not */
    #define RUNCMD      0x140C
/* Run command; normal mode set, lowest 2 prog. SPx switches set to ground,
   next 2 set to battery */
#else
    #define RUNCMD      0x140F
/* Run command; normal mode set, 4 prog. SPx switches set to battery */
#endif

/* define for MC33394 device SPI communication */
#define POWEROAKCMD 0x007F
/* everything is turned on except the Wake up capability */


/******************************************************************************/
/*              S T R U C T U R E S                                           */
/******************************************************************************/
typedef struct                  /* structure of analog[8].struc variable word */
{
    tU16 value : 10;    /* analog value of ADC */
    tU16 dumb  : 5;     /* reserved for future */
    tU16 mode  : 1;     /* mode configuration of ADC module */
        /* mode = 0 ... normal voltage measurement according to “range” value */
        /* mode = 1 ... current loop measurement */
} sAnalog;

typedef struct                  /* structure of analogConf byte variable */
{
    tU08 range : 2;     /* range configuration of ADC module */
    tU08 dumb  : 5;     /* reserved for future */
    tU08 accuracy : 1;  /* ADC module accuracy */
        /* accuracy = 0 ... 8 bit accuracy */
        /* accuracy = 1 ... 10 bit accuracy */
} sAnalogConf;

typedef union                  /* union for analog word variable  */
{
  tU16      word;       /* access whole word */
  struct                /* access byte at a time */
  {
    tU08    msb;
    tU08    lsb;
  } byte;
  sAnalog   struc;      /* access as declared in sAnalog structure */
} uAnalog;

typedef union                  /* union for digital variable word */
{
  tU16      word;       /* access whole word */
  struct                /* access byte at a time */
  {
    tU08    msb;
    tU08    lsb;
  } byte;
} uDigital;

typedef struct                  /* structure of the node information */
{
    tU08        nodeID;         /* node identification */
    uDigital    digitIn;        /* digital input values */
    uDigital    digitOut;       /* digital output values */
    uAnalog     analog[8];      /* union of analog value */
    sAnalogConf analogConf[8];  /* analog configuration structure */
} sNode;


/******************************************************************************/
/*        M C 9 S 1 2 D P 2 5 6   D e p e n d e n t   S t u f f               */
/******************************************************************************/
/* INTERRUPTS ENABLE / DISABLE function style macros */
#define archEnableInt()            {__asm CLI;}
#define archDisableInt()           {__asm SEI;}


/******************************************************************************/
/*       G E N E R I C   F U N C T I O N   S T Y L E   M A C R O S            */
/******************************************************************************/
/* Bit operation function style macros */
#define periphBitSet(Mask, Addr)        *(Addr) |= Mask
#define periphBitClear(Mask, Addr)      *(Addr) &= ~(Mask)
/* void periphBitChange(UWord16 Mask, volatile UWord16 * Addr); */
#define periphBitChange(Mask, Addr)     *(Addr) ^= Mask
/* bool periphBitTest(UWord16 Mask, volatile UWord16 * Addr); */
#define periphBitTest(Mask, Addr)       ( *(Addr) & (Mask) )


/******************************************************************************/
/*            A P P L I C A T I O N   C A N   D E F I N E S                   */
/******************************************************************************/
/* defines of “key” CAN 11bit long standard identifiers */
/* thise key identifiers are then enhanced by Node ID information */
/* note that this is a slight modification of msCAN driver functionality where
   identifiers are stored in ROM area */
#define CAN_KEYID_MSG_A1   0x0200
#define CAN_KEYID_MSG_A2   0x0240
#define CAN_KEYID_MSG_B    0x0100
#define CAN_KEYID_MSG_C    0x0040
#define CAN_KEYID_MSG_D    0x0402
#define CAN_KEYID_MSG_E    0x0405
```

```
/******************************************************************************/
/*              m s C A N   D R I V E R   D E F I N E S                       */
/******************************************************************************/
/* these two defines are necessary for proper msCAN driver operations */
#define HICROSS
#define MSCAN12


#endif
```

A.4 atd.c

```
/*******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: atd.c
*
* Description: Routines of the ATD module of the MC9S12DP256.
*
* Modules Included:
*       atd0Init(tU08 accuracy)
*       atd0Read()
*
*******************************************************************************/
#include “s12_regs.h”           /* register definition */

#include “CAN_slave.h”          /* project main header file */
#include “atd.h”
```

```
/******************************************************************************/
/*              G L O B A L   V A R I A B L E S                               */
/******************************************************************************/
extern volatile sNode node;     /* complete Node information */


/*******************************************************************************
*
* Module: void atd0Init(tU08 accuracy)
*
* Description: In this routine the initialization of ADC is done.
*       According to the parameter, is set the ADC accuracy to 8 or to 10 bits.
*
* Returns: None
*
* Global Data: None
*
* Arguments: accuracy is either ACCURACY_8BIT or ACCURACY_10BIT
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
void atd0Init(tU08 accuracy)
{
/* ATD configuration */
    atd0.atdctl2.bit.adpu = 1;      /* ADPU turns on ATD */
// interrupt
//  atd0.atdctl2.bit.ascie = 1;     /* ASCIE enables interrupt of the SCF */
    atd0.atdctl2.bit.affc = 1;
    /* AFFC turns on fast clear mode of sequence complete flag (SCF) */

    periphBitSet(S8C | S4C | S2C | S1C, &atd0.atdctl3.byte);
    /* number of conversion is 8 per sequence */
    atd0.atdctl3.bit.frz = 1;
    /* when BDM breakpoint come, current convers. is finished then freezed */
    atd0.atdctl4.byte = 0x3;        /* set total divisor to 8
       on EVM,  module clock is 8MHz, so ATD conversion clock it 1MHz */

    if (accuracy == ACCURACY_8BIT)      /* if 8-bit resolution is selected */
        atd0.atdctl4.bit.res8 = 1;      /* RES8 bit is set */
    else atd0.atdctl4.bit.res8 = 0;     /* otherwise 10-bit resolution */

    periphBitSet(DJM | SCAN | MULT, &atd0.atdctl5.byte);
    /* DJM sets right justified mode, SCAN sets conversion to sequence
       continuously, MULT sets to sample accros many channels */
}
```

```
/*******************************************************************************
*
* Module: void atd0Read(void)
*
* Description: This routine reads 8 ADC result registers and saves them to
*       node.analog[] structure.
*       Note that Analog Input 0 is mapped to AN07 pin of MCU, Analog Input 1
*       to AN06 and so on.
*
* Returns: None
*
* Global Data: node.analog[] is to where to save the analog values
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
void atd0Read(void)
{
    tU08 i;

    while(periphBitTest(SCF, &atd0.atdstat0.byte) == 0);  /* wait for flag */
    /* it is neccesary to wait for a complete flag */

    for (i = 0; i < 8; i++)                         /* read values */
    {
        node.analog[i].struc.value = atd0.atddr[7 - i].word;
       /* write analog value, note that Analog Input 0 is mapped to AN07 pin
          of MCU, Analog Input 1 to AN06 and so on */
    }
}
```

A.5 atd.h

```
/*******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: atd.h
*
* Description: Header file for the atd.c file
*
* Modules Included: None
*
*******************************************************************************/
#ifndef _atd_H_
#define _atd_H_


/******************************************************************************/
/*                P R O T O T Y P E S                                         */
/******************************************************************************/
void atd0Init(tU08 accuracy);
void atd0Read(void);

/******************************************************************************/
/*              A T D   D E F I N E S                                         */
/******************************************************************************/
/* defines for ATD accuracy selection */
#define     ACCURACY_8BIT       0
#define     ACCURACY_10BIT      1

#endif
```

A.6 io.c

```
/*******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
```

```
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*********************************************************************************
*
* File Name: io.c
*
* Description: Routines of the PIM (Port Integration Module) & IO module of the
*       MC9S12DP256.
*
* Modules Included:
*       ioInit()
*       portJISR()
*
********************************************************************************/
#include “s12_regs.h”               /* register definition */

#include “CAN_slave.h”              /* project main header file */
#include “io.h”
#include “spi.h”


/*******************************************************************************/
/*              G L O B A L   V A R I A B L E S                                */
/*******************************************************************************/


/********************************************************************************
*
* Module: void ioInit(void)
*
* Description: In this routine the initialization of PIM / IO module is done.
*       It configures:
*           - port A, B, T, H for analog channels configuration and control
*           - port E for 4 chip select signals + 2 Short Fault Protect pins
*           - port K for 6bit DIP switch for Node ID user’s change
*           - port J for device reset signals for MC33298 & MC33884 + interrupt
*               signal from MC33884
*           - port P for 2bit DIP switch for CAN baudrate user’s change
*           - port S for SS_CAN (STB) signal for MC33388
*           - port M for EN signal for MC33388 and chip select signal of MC33394
*       see more in io.h
```

```
*
* Returns: None
*
* Global Data: None
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues:
*
*******************************************************************************/
void ioInit(void)
{
/* Pull-up control for ports A, B, E, K */
    /* PURC not changed since port E is set as output and pull-ups for port K
       are desirable */

/* PORT B configuration */
    periphBitClear(AIC1 | AIC0, &reg.portb.byte); /* set default - all zeroes */
    periphBitSet(AIC1 | AIC0, &reg.ddrb.byte);  /* set ctrl pins as outputs */

/* PORT H configuration */
    periphBitClear(AIC7 | AIC6, &pim.pth.byte); /* set default - all zeroes */
    periphBitSet(AIC7 | AIC6, &pim.ddrh.byte);  /* set ctrl pins as outputs */

/* PORT A configuration */
    periphBitClear(AIC5 | AIC4, &reg.porta.byte); /* set default - all zeroes */
    periphBitSet(AIC5 | AIC4, &reg.ddra.byte);  /* set ctrl pins as outputs */

/* PORT T configuration */
    periphBitClear(AIC3 | AIC2, &pim.ptt.byte);  /* set default - all zeroes */
    periphBitSet(AIC3 | AIC2, &pim.ddrt.byte);  /* set ctrl pins as outputs */

/* PORT E configuration */
    reg.pear.bit.neclk = 1;
    /* set pin PE4 as GPIO,  rest of port E pins are GPIO after reset */
    periphBitSet(CSB0 | CSB1 | CSB2 | CSB3, &reg.porte.byte);
    /* set values of ChipSelect pins to 1 */
    periphBitSet(SFPD0 | SFPD1, &reg.porte.byte);
    /* set values of Short Fault Protect pins to 1 - output(s) will remain
       “on” in a current limited mode of operation */
    periphBitSet(SFPD0 | SFPD1 | CSB0 | CSB1 | CSB2 | CSB3, &reg.ddre.byte);
    /* set pins as outputs */

/* PORT K configuration */
    page.ddrk.byte = 0xC0;   /* lower 6 bits as inputs for Node ID DIP switch */

/* PORT J configuration */
    periphBitSet(RESIN | RESOUT, &pim.ptj.byte);
    /* set both reset signal to inactive state (to logical 1) */
    periphBitSet(RESIN | RESOUT, &pim.ddrj.byte);
    /* both resets set as outputs, interrupt signal from MC33884 as input */
    /* RDRJ not changed, both outputs with full drive strength */
    pim.perj.bit.perj0 = 1;
    /* pull-up/down device is enabled for interrupt signal from MC33884  */
    /* PPSJ not changed, falling edge set for interrupt signal from MC33884 */
//    pim.piej.bit.piej0 = 1;
    /* interrupt is disabled for the interrupt signal from MC33884 */

/* PORT P configuration */
    /* DDRP not changed, since BRD0 and BDR1 set as inputs after reset */
    /* RDRP not changed, settings is ignored when confugured as input */
    periphBitSet(DIP_BDR, &pim.perp.byte);
    /* enable pull-up or pull-down device for DIP switch bits  */
    /* PPSJ not changed, pull-up is set after reset */
    /* PIEP not changed, no interrupts */

/* PORT S configuration */
    pim.pts.bit.pts7 = 1;   /* set bit - SS_CAN (STB) signal for MC33388 */
    pim.ddrs.bit.ddrs7 = 1; /* set PE7 bit as output */

/* PORT M configuration */
    pim.ptm.bit.ptm7 = 1;   /* set bit - it is EN signal for MC33388 */
    pim.ddrm.bit.ddrm7 = 1; /* set PE7 bit as output */

    clearCSB4();            /* clear bit - it is chip select for MC33394 */
    /* note that non-active state of CSB for this device is LOW! */
    pim.ddrm.bit.ddrm2 = 1; /* set PE2 bit as output */
}
```

A.7 io.h

```
/*******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: io.h
*
* Description: Header file for the io.c file
*
* Modules Included: None
*
*******************************************************************************/
#ifndef _io_H_
#define _io_H_


/******************************************************************************/
/*                P R O T O T Y P E S                                         */
/******************************************************************************/
void ioInit(void);


/******************************************************************************/
/*                GPIO mapping                                                */
/******************************************************************************/
/* for SPI communication with MC33298 & MC33884 devices */
#define CSB0    PTE2
#define CSB1    PTE4
#define CSB2    PTE6
#define CSB3    PTE7
#define SFPD0   PTE3
#define SFPD1   PTE5

/* reset signals for MC33298 & MC33884 + interrupt signal from MC33884 */
#define INTB    PTJ0
#define RESIN   PTJ6
#define RESOUT  PTJ7

/* for SPI communication with MC33394 Power Oak device */
#define CSB4    PTM2

/* DIP Switch mapping */
#define DIP_NODEID      (PTK0 | PTK1 | PTK2 | PTK3 | PTK4 | PTK5)
#define DIP_BDR         (PTP0 | PTP1)

/* control pins of ADC convertor - Analog Input Control pins */
/* pin mapping difference due to HW Rev. 01 */
#define AIC0    PTB5 | PTB4
#define AIC1    PTB1 | PTB0
#define AIC2    PTT5 | PTT4
```

```
#define AIC3    PTT1 | PTT0
#define AIC4    PTA5 | PTA4
#define AIC5    PTA1 | PTA0
#define AIC6    PTH5 | PTH4
#define AIC7    PTH1 | PTH0

/* signal pins of ADC convertor - Analog Input Signalization pins */
#define AIS0    PTB7
#define AIS1    PTB3
#define AIS2    PTT7
#define AIS3    PTT3
#define AIS4    PTA7
#define AIS5    PTA3
#define AIS6    PTH7
#define AIS7    PTH3

/* shorcuts for control pins of ADC convertor - Analog Input Control pins */
/* pin mapping difference due to HW Rev. 01 */
#define pAIC0_0     reg.portb.bit.ptb4
#define pAIC0_1     reg.portb.bit.ptb5
#define pAIC1_0     reg.portb.bit.ptb0
#define pAIC1_1     reg.portb.bit.ptb1
#define pAIC2_0     pim.ptt.bit.ptt4
#define pAIC2_1     pim.ptt.bit.ptt5
#define pAIC3_0     pim.ptt.bit.ptt0
#define pAIC3_1     pim.ptt.bit.ptt1
#define pAIC4_0     reg.porta.bit.pta4
#define pAIC4_1     reg.porta.bit.pta5
#define pAIC5_0     reg.porta.bit.pta0
#define pAIC5_1     reg.porta.bit.pta1
#define pAIC6_0     pim.pth.bit.pth4
#define pAIC6_1     pim.pth.bit.pth5
#define pAIC7_0     pim.pth.bit.pth0
#define pAIC7_1     pim.pth.bit.pth1

/* shorcuts for signal pins of ADC convertor - Analog Input Signalization pins */
#define pAIS0   reg.portb.bit.ptb7
#define pAIS1   reg.portb.bit.ptb3
#define pAIS2   pim.ptt.bit.ptt7
#define pAIS3   pim.ptt.bit.ptt3
#define pAIS4   reg.porta.bit.pta7
#define pAIS5   reg.porta.bit.pta3
#define pAIS6   pim.pth.bit.pth7
#define pAIS7   pim.pth.bit.pth3


/******************************************************************************/
/*                Function style macros                                       */
/******************************************************************************/
/* chip select signal control for MC33298 & MC33884 */
/* all there Chip select signal are active low */
```

```
#define setCSB0()       periphBitSet(CSB0, &reg.porte.byte)
#define setCSB1()       periphBitSet(CSB1, &reg.porte.byte)
#define setCSB2()       periphBitSet(CSB2, &reg.porte.byte)
#define setCSB3()       periphBitSet(CSB3, &reg.porte.byte)
#define clearCSB0()     periphBitClear(CSB0, &reg.porte.byte)
#define clearCSB1()     periphBitClear(CSB1, &reg.porte.byte)
#define clearCSB2()     periphBitClear(CSB2, &reg.porte.byte)
#define clearCSB3()     periphBitClear(CSB3, &reg.porte.byte)

/* chip select signal control for MC33394 */
/* this Chip select signal is active high */
#define setCSB4()       periphBitSet(CSB4, &pim.ptm.byte)
#define clearCSB4()     periphBitClear(CSB4, &pim.ptm.byte)

/* reset control for MC33298 & MC33884 devices */
#define resetInputIO()  periphBitClear(RESIN, &pim.ptj.byte);   \
                        { asm NOP; } { asm NOP; } { asm NOP; }  \
                        periphBitSet(RESIN, &pim.ptj.byte)
#define resetOutputIO() periphBitClear(RESOUT, &pim.ptj.byte);  \
                        { asm NOP; } { asm NOP; } { asm NOP; }  \
                        periphBitSet(RESOUT, &pim.ptj.byte)

/* DIP Switch reading */
#define readDipNodeID() periphBitTest(DIP_NODEID, &page.portk.byte)
    /* read value from the DIP switch dedicated for Node identification */
#define readDipBdr()    periphBitTest(DIP_BDR, &pim.ptp.byte)
    /* read value from the DIP switch dedicated for CAN baudrate speed */

#endif
```

A.8 rti.c

```
/******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*******************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA "AS IS" AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: rti.c
*
* Description: Routines of the RTI module of the MC9S12DP256.
*
* Modules Included:
*       rtiInit()
*       rtiISR()
*
*******************************************************************************/
#include “s12_regs.h”       /* register definition */

#include “CAN_slave.h”      /* project main header file */
#include “rti.h”
#include “atd.h”

#include “msCANstd.h”       /* msCAN module */
#include “msCANdrv.h”


/******************************************************************************/
/*              G L O B A L   V A R I A B L E S                              */
/******************************************************************************/
extern volatile sNode node;             /* complete Node information */


/*******************************************************************************
*
* Module: void rtiInit(void)
*
* Description: In this routine the initialization of RTI is done.
*       When running with 4MHz crystal, it interrupts 15.25 times per second.
*       When running with 16MHz crystal, it interrupts 15.25 times per second
*       as well.
*
* Returns: None
*
* Global Data:
*       OSC_16MHZ is a symbolic constant, when defined, 16MHz crystal is
*           connected to board
*       OSC_4MHZ is a symbolic constant, when defined 4MHz crystal is
*           connected to board
*
* Arguments: None
*
* Range Issues: None
```

```
*
* Special Issues:
*
*******************************************************************************/
void rtiInit(void)
{
#ifdef OSC_4MHZ
    crg.rtictl.byte = 0x73;     /* real time interrupt 15.25 times per second */
#endif
#ifdef OSC_16MHZ
    crg.rtictl.byte = 0x7F;     /* real time interrupt 15.25 times per second */
#endif

    crg.crgint.bit.rtie = 1;    /* real time interrupt enable */
}


/*******************************************************************************
*
* Module: void rtiISR(void)
*
* Description: This routine is the interrupt service routine of the RTI module,
*       it interrupts the program 15.25 times per second.
*       It it used for CAN message sending (both analog and digital status msg)
*
*       When “black box” application is enabled, it is used for generation of
*       two CAN analog status messages 15.25 times per second + CAN digital
*       status message once per second approximately.
*       When “black box” application is disabled, it sends two CAN analog status
*       + one CAN digital messages once per second (distributed throught the
*       time)
*
* Returns: None
*
* Global Data:
*       BLACK_BOX is a symbolic constant, if defined the “black box” demo
*           application is enabled
*       node.analog
*       node.digitIn
*       node.digitOut
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues:
*       ADC conversion is done in pooling fashion.
*
*******************************************************************************/
#pragma TRAP_PROC
void rtiISR(void)
```

```
{
    tU08 tmp;
    tU08 sendData[9];       /* pass data to CAN Tx routine */
    static count = 0;       /* routine counter */

    crg.crgflg.bit.rtif = 1;    /* clear flag of real time interrupt */
    count++;

    atd0Read();             /* perform analog to digital conversion */

#if 0
    while (sci0.scisr1.bit.tdre == 0);  /* SCI testing transmission */
    tmp = sci0.scisr1.byte;
    sci0.scidrl.byte = 0x41;
#endif

#ifdef BLACK_BOX
        set1stAnalogData();                 /* store desired data to sendData */
        tmp = CAN_LoadMB(3, sendData, 0);   /* load buf */
        tmp = CAN_TransmitMB(3, 0);         /* send buf */
        set2ndAnalogData();                 /* store desired data to sendData */
        tmp = CAN_LoadMB(4, sendData, 0);   /* load buf */
        tmp = CAN_TransmitMB(4, 0);         /* send buf */
#else
    if (count == 13)        /* analog status, part 1 */
    {
        set1stAnalogData();                 /* store desired data to sendData */
        tmp = CAN_LoadMB(3, sendData, 0);   /* load buf */
        tmp = CAN_TransmitMB(3, 0);         /* send buf */
    }
    if (count == 14)        /* analog status, part 2 */
    {
        set2ndAnalogData();                 /* store desired data to sendData */
        tmp = CAN_LoadMB(4, sendData, 0);   /* load buf */
        tmp = CAN_TransmitMB(4, 0);         /* send buf */
    }
 #endif

    if (count == 15)
    {
        count = 0;
        setDigitData();                     /* store desired data to sendData */
        tmp = CAN_LoadMB(2, sendData, 0);   /* load buf */
        tmp = CAN_TransmitMB(2, 0);         /* send buf */
    }
}
```

A.9 rti.h

```
/*******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: rti.h
*
* Description: Header file for the rti.c file
*
* Modules Included: None
*
*******************************************************************************/
#ifndef _rti_H_
#define _rti_H_

#include “io.h”

/******************************************************************************/
/*                P R O T O T Y P E S                                         */
/******************************************************************************/
void rtiInit(void);
void rtiISR(void);


/******************************************************************************/
/*                RTI Function style macros                                   */
/******************************************************************************/
#define set1stAnalogData() sendData[0] = 8;     \
                         for (tmp = 0; tmp < 4; tmp ++)  \
                         {  sendData[2 * tmp + 1] = node.analog[tmp].byte.lsb; \
                            sendData[2 * tmp + 2] = node.analog[tmp].byte.msb; }
    /* prepare first 4 analog variables for CAN transmission */
```

```
#define set2ndAnalogData() sendData[0] = 8;    \
                          for (tmp = 4; tmp < 8; tmp ++)  \
                          {  sendData[2 * tmp - 7] = node.analog[tmp].byte.lsb; \
                             sendData[2 * tmp - 6] = node.analog[tmp].byte.msb; }
    /* prepare second 4 analog variables for CAN transmission */

#define setDigitData()    sendData[0] = 4;    \
                          sendData[1] = node.digitIn.byte.lsb;   \
                          sendData[2] = node.digitIn.byte.msb;   \
                          sendData[3] = node.digitOut.byte.lsb;  \
                          sendData[4] = node.digitOut.byte.msb
    /* prepare digital variables for CAN transmission */


#endif
```

A.10 sci.c

```
/********************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: sci.c
*
* Description: Routines of the SCI module of the MC9S12DP256.
*
* Modules Included:
*       sci0Init()
*
```

```
*******************************************************************************/
#include “s12_regs.h”           /* register definition */

#include “CAN_slave.h”          /* project main header file */
#include “sci.h”


/*******************************************************************************
*
* Module: void sci0Init(void)
*
* Description: In this routine the initialization of SCI is done.
*       When running with 16 MHz crystal, the SCI baudrate is set to 38.400pbs.
*       When running with 4 MHz crystal, the SCI baudrate is set to 9.600pbs.
*
* Returns: None
*
* Global Data:
*       OSC_16MHZ is a symbolic constant, when defined, 16MHz crystal is
*           connected to board
*       OSC_4MHZ is a symbolic constant, when defined 4MHz crystal is
*           connected to board
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
void sci0Init(void)
{
    tU08 tmp;

#ifdef OSC_16MHZ
    /* Module Clock with EVB = 16 / 2 MHz */
    sci0.scibd.word = 0x0D;    /* baudrate is set to 38.400 */
    /* 0x0034: BR = 9.600Bd, 0x001A: BR = 19.200Bd, 0x000D: BR = 38.400Bd */
#endif
#ifdef OSC_4MHZ
    /* Module Clock on module = 4 / 2 MHz */
    sci0.scibd.word = 0x0D;    /* baudrate is set to 9.600 */
    /* 0x0034: BR = 2.400Bd, 0x001A: BR = 4.800Bd, 0x000D: BR = 9.600Bd */
#endif

    sci0.scicr2.byte = TE | RE; /* set TE, RE */
    tmp = sci0.scisr1.byte;     /* clear Status Register */
}
```

A.11 sci.h

```
/*******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: sci.h
*
* Description: Header file for the sci.c file
*
* Modules Included: None
*
*******************************************************************************/
#ifndef _sci_H_
#define _sci_H_


/******************************************************************************/
/*                P R O T O T Y P E S                                         */
/******************************************************************************/
void sci0Init(void);


/******************************************************************************/
/*                SCI Function style macros                                   */
/******************************************************************************/
#define sci0Read()                 sci0.scidrl
#define sci0Write(x)               { sci0.scidrl = x; }


#endif
```

A.12 spi.c

```
/*******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: spi.c
*
* Description: Routines of the SPI module of the MC9S12DP256.
*       SPI format of communication is used for the configuration with the
*       following devices:
*       PC33394 - Switch Mode Power Supply with Multiple Linear Regulators and
*           High speed CAN Transceiver
*       MC33884 - Switch Monitor Interface (digital inputs)
*       MC33298 - Octal Serial Switch (digital outputs)
*
* Modules Included:
*       spi0Init()
*       tU08 spi0TxByte ( tU08 chipSelect, tU08 byte)
*       tU16 spi0TxWord (tU08 chipSelect, tU16 cmd)
*
*******************************************************************************/
#include “s12_regs.h”               /* register definition */

#include “CAN_slave.h”              /* project main header file */
#include “spi.h”
#include “io.h”


/*******************************************************************************
*
* Module: void spi0Init(void)
*
```

```
* Description: The SPI channel is used for communication with MC33884, MC33298
*       and Power Oak MC33394. (Power Oak is used when FAST_CAN_ENABLE symbolic
*       constant is defined, otherwise the MC33388D device is used instead.)
*       This routine configures the SPI communication parameters.
*       Finally it configures:
*           - Switch Monitor Interface MC33884 into operation.
*           - Power Oak MC33394 device (if enabled)
*
* Returns: None
*
* Global Data: None
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: GPIO initialization has to be done before
*
*******************************************************************************/
void spi0Init(void)
{
    tU08 tmp;

    spi0.spicr1.bit.lsbf  = 0;      /* lsb first enable bit */
                                    /* msb bit is transferred first */
    spi0.spicr1.bit.ssoe  = 0;      /* slave select output enable */
                                    /* slave select output is not enabled */
    spi0.spicr1.bit.cpha  = 1;      /* SPI clock phase bit */
                                    /* first SCLK edge issued at the beginning
                                       of the 8-cycle transfer operation */
    spi0.spicr1.bit.cpol  = 0;      /* clock polarity bit */
                                    /* serial clock (SCK) active in high, SCK
                                       idles low */
    spi0.spicr1.bit.mstr  = 1;      /* master/slave mode select bit */
                                    /* Master mode selected */
    spi0.spicr1.bit.sptie = 0;      /* transmit interrupt enable bit */
                                    /* transmit interrupt disabled, SPI
                                       communication done in pooling  style */
    spi0.spicr1.bit.spe   = 1;      /* spi enable bit */
                                    /* enable SPI, SPI port pins are dedicated
                                       to SPI module */
    spi0.spicr1.bit.spie  = 0;      /* spi interrupt enable bit */
                                    /* SPI interrupt disabled */

    spi0.spicr2.bit.spc0    = 0;    /* serial pin control 0 bit */
                                    /* no bidirectional pin configuration of
                                       the SPI */
    spi0.spicr2.bit.spiswai = 0;    /* SPI stop in wait mode bit */
                                    /* SCLK operates normally in wait mode */
    spi0.spicr2.bit.bidiroe = 0;    /* bi-directional mode output enable bit */
                                    /* output buffer disable in bidirectional
```

```
                                       mode */
    spi0.spicr2.bit.modfen  = 0;    /* mode fault enable bit */
                                    /* disable the MODF error */

    /* divider is set to 8, so SCLK is 1MHz for 8MHz Module Clk */
//    spi0.spibr.bit.spr  = 2;        /* baud rate selection */
//    spi0.spibr.bit.sppr = 0;        /* baud rate pre-selection */

    /* in order to run the SCLK on 4MHz while 16MHz crystal is connected
      (thus 8 MHz Module CLK), SPI module clock divisor has to be 2 */
    /* divider is set to 2, so SCLK is 4MHz for 8MHz Module Clk */
    spi0.spibr.bit.spr  = 0;        /* baud rate selection */
    spi0.spibr.bit.sspr = 0;        /* baud rate pre-selection */

    /* initialize both MC33884 chips for the first time */
    tmp = spi0TxWord(CSB2, TRISTATECMD);  /* first, tri-state command */
    tmp = spi0TxWord(CSB2, METALLICCMD);  /* then, metallic command */
    tmp = spi0TxWord(CSB2, RUNCMD);       /* finally, run command */

    tmp = spi0TxWord(CSB3, TRISTATECMD);
    tmp = spi0TxWord(CSB3, METALLICCMD);
    tmp = spi0TxWord(CSB3, RUNCMD);

#ifdef FAST_CAN_ENABLE
    /* initialize Power Oak MC33394 device for the first time */
    tmp = spi0TxWord(CSB4, POWEROAKCMD);  /* configure Power Oak */
#endif
}


/*******************************************************************************
*
* Module: tU08 spi0TxByte (tU08 chipSelect, tU08 byte)
*
* Description: This is the SPI communication function. It transmit one byte via
*       SPI and pass the received one as an argument.
*       This routine is used for the MC33298 device communication
*
* Returns: tmp as the SPI received byte
*
* Global Data: None
*
* Arguments:
*       chipSelect in order to choose the desired device. Possible velues
*           are CSB0 and CSB1
*       byte as the value to be transmit
*
* Range Issues: possible values for chipSelect are CSB0 and CSB1 only
*
* Special Issues: Note that SPI format is different for each device. Proper
*       setting is done by calling proper function style macro (see spi.h
*       for more)
*
******************************************************************************/
tU08 spi0TxByte (tU08 chipSelect, tU08 byte)
{
    tU08 tmp;

    setSPIForOutputs(); /* configure SPI format properly */

    while (spi0.spisr.bit.sptef == 0);          /* while Tx reg not empty */
    if (chipSelect == CSB0) clearCSB0();        /* set "chip select" */
    else if (chipSelect == CSB1) clearCSB1();
    spi0Write(byte);                            /* write value */
    while (spi0.spisr.bit.spif == 0);           /* while Rx reg not empty */
    tmp = spi0Read();                           /* store status byte */
    while (spi0.spisr.bit.sptef == 0);          /* while Tx reg not empty */
    if (chipSelect == CSB0) setCSB0();          /* unset "chip select" */
    else if (chipSelect == CSB1) setCSB1();
    return (tmp);
}


/******************************************************************************
*
* Module: tU16 spi0TxWord (tU08 chipSelect, tU16 cmd)
*
* Description: This is the SPI communication function. It transmit one word via
*       SPI and pass the received one as an argument.
*       This routine is used for the MC33884 and MC33394 "Power Oak" device
*       communication.
*       Note that Power Oak device has CSB active in high while MC33884 device
*       in low.
*
* Returns: tmp as the SPI received word
*
* Global Data: None
*
* Arguments:
*       chipSelect in order to choose the desired device. Possible velues
*           are CSB2 and CSB3 only (MC33884) and CSB4 for MC33394 Power Oak
*       cmd as the value to be transmit
*
* Range Issues: possible values for chipSelect are CSB2 and CSB3 only (MC33884)
*       and CSB4 for MC33394 Power Oak
*
* Special Issues: Note that SPI format is different for each device. Proper
*       setting is done by calling proper function style macro (see spi.h
*       for more)
*
******************************************************************************/
tU16 spi0TxWord (tU08 chipSelect, tU16 cmd)
```

```
{
    tU16 tmp;

    if (chipSelect == CSB4)     /* when Power Oak communication desired */
    {
        setSPIForPowerOak(); /* configure SPI format properly for MC33394 */

        while (spi0.spisr.bit.sptef == 0);      /* while Tx reg not empty */
        setCSB4();                              /* set “chip select” */

        spi0Write(cmd & 0x00FF);                /* send lower part of command */
        while (spi0.spisr.bit.spif == 0);       /* while Rx reg not empty */
        tmp = spi0Read();                       /* store lower byte of status */

        while (spi0.spisr.bit.sptef == 0);      /* while Tx reg not empty */
        spi0Write((cmd >> 8) & 0x00FF);         /* send higher part of command*/
        while (spi0.spisr.bit.spif == 0);       /* while Rx reg not empty */
        tmp |= spi0Read() << 8;                 /* store higher byte of status*/
        while (spi0.spisr.bit.sptef == 0);      /* while Tx reg not empty */
        clearCSB4();                            /* unset “chip select” */

        return (tmp);  /* return received word */
    }
    else if ((chipSelect == CSB2) || (chipSelect == CSB3))  /* digital inputs */
    {
        setSPIForInputs(); /* configure SPI format properly for MC33884 */

        while (spi0.spisr.bit.sptef == 0);      /* while Tx reg not empty */
        if (chipSelect == CSB2) clearCSB2();    /* set “chip select” */
        else if (chipSelect == CSB3) clearCSB3();

        spi0Write((cmd >> 8) & 0x00FF);         /* send a command to device */
        while (spi0.spisr.bit.spif == 0);       /* while Rx reg not empty */
        tmp = spi0Read() << 8;                  /* store higher byte of status */

        while (spi0.spisr.bit.sptef == 0);      /* while Tx reg not empty */
        spi0Write(cmd & 0x00FF);                /* send a command to device */
        while (spi0.spisr.bit.spif == 0);       /* while Rx reg not empty */
        tmp |= spi0Read();                      /* store status-lower byte */
        while (spi0.spisr.bit.sptef == 0);      /* while Tx reg not empty */
        if (chipSelect == CSB2) setCSB2();      /* unset “chip select” */
        else if (chipSelect == CSB3) setCSB3();

        return (tmp >> 2);  /* shift since bits SB1 and SB2 are not in use */
    }
}
```

A.13 spi.h

```
/*******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA "AS IS" AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: spi.h
*
* Description: Header file for the spi.c file
*
* Modules Included: None
*
*******************************************************************************/
#ifndef _spi_H_
#define _spi_H_


/******************************************************************************/
/*                P R O T O T Y P E S                                         */
/******************************************************************************/
void spi0Init(void);
tU08 spi0TxByte (tU08 chipSelect, tU08 byte);
tU16 spi0TxWord (tU08 chipSelect, tU16 cmd);


/******************************************************************************/
/*                SPI Function style macros                                   */
/******************************************************************************/
#define spi0Read()              spi0.spidr.byte
#define spi0Write(x)            { spi0.spidr.byte = x; }


/******************************************************************************/
```

```
/*                Function style macros                                       */
/******************************************************************************/
/* re-initialization of the SPI communication format for different devices:
   MC33298 (digital outputs):
        bit LSBF = 0    (lsb first enable)
        bit CPHA = 1    (clock phase bit, first SCLK edge at begin)
   MC33884 (digital inputs):
        bit LSBF = 0    (lsb first enable)
        bit CPHA = 0    (clock phase bit, first SCLK edge at begin)
   MC33394 (Power Oak)
        bit LSBF = 1    (lsb first enable)
        bit CPHA = 0    (clock phase bit, first SCLK edge at begin) */

#define setSPIForInputs()   spi0.spicr1.bit.cpha = 0;   \
                            spi0.spicr1.bit.lsbf = 0
#define setSPIForOutputs()  spi0.spicr1.bit.cpha = 1;   \
                            spi0.spicr1.bit.lsbf = 0
#define setSPIForPowerOak() spi0.spicr1.bit.cpha = 0;   \
                            spi0.spicr1.bit.lsbf = 1

#endif
```

A.14 s12_regs.c

```
/*******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: s12_regs.c
*
```

```
* Description: Periphery module allocation
*
* Modules Included: None
*
*******************************************************************************/
#include “s12_common.h”
#include “s12_atd.h”
#include “s12_bdlc.h”
#include “s12_crg.h”
#include “s12_eeprom.h”
#include “s12_flash.h”
#include “s12_iic.h”
#include “s12_mscan.h”
#include “s12_page.h”
#include “s12_pim.h”
#include “s12_pwm.h”
#include “s12_register.h”
#include “s12_sci.h”
#include “s12_spi.h”
#include “s12_template.h”
#include “s12_timer.h”

#pragma DATA_SEG SHORT REG_REG
tREGISTER reg;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG SHORT PAGE_REG
tPAGE page;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG SHORT CRG_REG
tCRG crg;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG PIM_REG
tPIM pim;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG SHORT TIM_REG
tTIMER tim;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG SHORT ATD0_REG
tATD atd0;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG SHORT PWM_REG
tPWM pwm;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG SHORT SCI0Regs
```

```
tSCI sci0;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG SHORT SCI1Regs
tSCI sci1;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG SHORT SPI0Regs
tSPI spi0;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG SHORT SPI1Regs
tSPI spi1;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG SHORT SPI2Regs
tSPI spi2;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG SHORT IIC_REG
tIIC iic;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG SHORT BDLC_REG
tBDLC bdlc;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG FLSH_REG
tFLASH flash;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG EPRM_REG
tEEPROM eeprom;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG ATD1_REG
tATD atd1;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG CAN0_REG
tMSCAN can0;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG CAN1_REG
tMSCAN can1;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG CAN2_REG
tMSCAN can2;
#pragma DATA_SEG DEFAULT
```

```
#pragma DATA_SEG CAN3_REG
tMSCAN can3;
#pragma DATA_SEG DEFAULT

#pragma DATA_SEG CAN4_REG
tMSCAN can4;
#pragma DATA_SEG DEFAULT
```

A.15 s12_regs.h

```
/*******************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
* EVENT SHALL MOTOROLA OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
********************************************************************************
*
* File Name: s12_regs.h
*
* Description: Periphery module allocation
*
* Modules Included: None
*
*******************************************************************************/
#include “s12_common.h”
#include “s12_atd.h”
#include “s12_bdlc.h”
#include “s12_crg.h”
#include “s12_eeprom.h”
#include “s12_flash.h”
#include “s12_iic.h”
#include “s12_mscan.h”
#include “s12_page.h”
```

```
#include “s12_pim.h”
#include “s12_pwm.h”
#include “s12_register.h”
#include “s12_sci.h”
#include “s12_spi.h”
#include “s12_template.h”
#include “s12_timer.h”


extern tREGISTER reg;

extern tPAGE page;

extern tCRG crg;

extern tPIM pim;

extern tTIMER tim;

extern tATD atd0;

extern tPWM pwm;

extern tSCI sci0;

extern tSCI sci1;

extern tSPI spi0;

extern tSPI spi1;

extern tSPI spi2;

extern tIIC iic;

extern tBDLC bdlc;

extern tFLASH flash;

extern tEEPROM eeprom;

extern tATD atd1;

extern tMSCAN can0;

extern tMSCAN can1;

extern tMSCAN can2;

extern tMSCAN can3;
extern tMSCAN can4;
```

A.16 MC9S12DP256_RAM.prm

```
NAMES
    msCANs12drv.o
END

SECTIONS
    REG_RG0 = NO_INIT 0x0000 TO 0x002F;
    PAGE_RG0= NO_INIT 0x0030 TO 0x0033;
    CRG_RG0 = NO_INIT 0x0034 TO 0x003F;
    TIM_RG0 = NO_INIT 0x0040 TO 0x007F;
    ATD_RG0 = NO_INIT 0x0080 TO 0x009F;
    PWM_RG0 = NO_INIT 0x00A0 TO 0x00C7;
    SCI_RG0 = NO_INIT 0x00C8 TO 0x00CF;
    SCI_RG1 = NO_INIT 0x00D0 TO 0x00D7;
    SPI_RG0 = NO_INIT 0x00D8 TO 0x00DF;
    IIC_RG0 = NO_INIT 0x00E0 TO 0x00E7;
    BDL_RG0 = NO_INIT 0x00E8 TO 0x00EF;
    SPI_RG1 = NO_INIT 0x00F0 TO 0x00F7;
    SPI_RG2 = NO_INIT 0x00F8 TO 0x00FF;

// from here on can not be short definitions
    FSH_RG0 = NO_INIT 0x0100 TO 0x010F;
    EE2_RG0 = NO_INIT 0x0110 TO 0x011B;
    ATD_RG1 = NO_INIT 0x0120 TO 0x013F;
    CAN_RG0 = NO_INIT 0x0140 TO 0x017F;
    CAN_RG1 = NO_INIT 0x0180 TO 0x01BF;
    CAN_RG2 = NO_INIT 0x01C0 TO 0x01FF;
    CAN_RG3 = NO_INIT 0x0200 TO 0x023F;
    PIM_RG0 = NO_INIT 0x0240 TO 0x026F;
    CAN_RG4 = NO_INIT 0x0280 TO 0x02BF;

    MY_RAM        = READ_WRITE 0x1010 TO 0x1FFF;
    MY_PSEUDO_ROM = READ_ONLY  0x2000 TO 0x3FFF;

    MSCAN0_START = NO_INIT  0x0140 TO 0x0140;
END

PLACEMENT
    _PRESTART, STARTUP,
    ROM_VAR, STRINGS,
    NON_BANKED,DEFAULT_ROM,
    COPY                  INTO  MY_PSEUDO_ROM;
    DEFAULT_RAM           INTO  MY_RAM;

    CAN0_REG              INTO  CAN_RG0;
    CAN1_REG              INTO  CAN_RG1;
    CAN2_REG              INTO  CAN_RG2;
    CAN3_REG              INTO  CAN_RG3;
    CAN4_REG              INTO  CAN_RG4;
    FLSH_REG              INTO  FSH_RG0;
    EPRM_REG              INTO  EE2_RG0;
    BDLC_REG              INTO  BDL_RG0;
    IIC_REG               INTO  IIC_RG0;
    TIM_REG               INTO  TIM_RG0;
    PAGE_REG              INTO  PAGE_RG0;
    SCI0Regs              INTO  SCI_RG0;
    SCI1Regs              INTO  SCI_RG1;
    SPI0Regs              INTO  SPI_RG0;
    SPI1Regs              INTO  SPI_RG1;
    SPI2Regs              INTO  SPI_RG2;
    PWM_REG               INTO  PWM_RG0;
    REG_REG               INTO  REG_RG0;
    CRG_REG               INTO  CRG_RG0;
    ATD0_REG              INTO  ATD_RG0;
    ATD1_REG              INTO  ATD_RG1;
    PIM_REG               INTO  PIM_RG0;

    MSCAN0                INTO  MSCAN0_START;

END

STACKSIZE 0x100

VECTOR 0  _Startup
VECTOR ADDRESS 0xFFF0     rtiISR      /* real time interrupt service routine */
VECTOR ADDRESS 0xFFB0     CAN0_TransmitISR    /* CAN0 Tx */
VECTOR ADDRESS 0xFFB2     CAN0_ReceiveISR     /* CAN0 Rx */
VECTOR ADDRESS 0xFFB6     CAN0_WakeupISR      /* CAN0 Wake-up */

/*  Interrupt Vector Table */
/*  0xFF8C:8D PWM Emergency Shutdown
    0xFF8E:8F Port P Interrupt
    0xFF90:91 MSCAN 4 transmit
    0xFF92:93 MSCAN 4 receive
    0xFF94:95 MSCAN 4 errors
    0xFF96:97 MSCAN 4 wake- up
    0xFF98:99 MSCAN 3 transmit
    0xFF9A:9B MSCAN 3 receive
    0xFF9C:9D MSCAN 3 errors
    0xFF9E:9F MSCAN 3 wake- up
    0xFFA0:A1 MSCAN 2 transmit
    0xFFA2:A3 MSCAN 2 receive
    0xFFA4:A5 MSCAN 2 errors
    0xFFA6:A7 MSCAN 2 wake-up
    0xFFA8:A9 MSCAN 1 transmit
    0xFFAA:AB MSCAN 1 receive
    0xFFAC:AD MSCAN 1 errors
    0xFFAE:AF MSCAN 1 wake-up
    0xFFB0:B1 MSCAN 0 transmit
    0xFFB2:B3 MSCAN 0 receive
    0xFFB4:B5 MSCAN 0 errors
    0xFFB6:B7 MSCAN 0 wake-up
    0xFFB8:B9 FLASH
    0xFFBA:BB EEPROM
    0xFFBC:BD SPI2
    0xFFBE:BF SPI1
    0xFFC0:C1 IIC Bus
    0xFFC2:C3 DLC
    0xFFC4:C5 SCME
    0xFFC6:C7 CRG lock
    0xFFC8:C9 Pulse Accumulator B Overflow
    0xFFCA:CB Modulus Down Counter underflow
    0xFFCC:CD Port H
    0xFFCE:CF Port J
    0xFFD0:D1 ATD1
    0xFFD2:D3 ATD0
    0xFFD4:D5 SCI1
    0xFFD6:D7 SCI0
    0xFFD8:D9 SPI0
    0xFFDA:DB Pulse accumulator input edge
    0xFFDC:DD Pulse accumulator A overflow
    0xFFDE:DF Timer overflow
    0xFFE0:E1 Timer channel 7
    0xFFE2:E3 Timer channel 6
    0xFFE4:E5 Timer channel 5
    0xFFE6:E7 Timer channel 4
    0xFFE8:E9 Timer channel 3
    0xFFEA:EB Timer channel 2
    0xFFEC:ED Timer channel 1
    0xFFEE:EF Timer channel 0
    0xFFF0:F1 RTI - Real time interrupt
    0xFFF2:F3 IRQ
    0xFFF4:F5 XIRQ
    0xFFF6:F7 SWI
    0xFFF8:F9 Unimplemented instruction trap
    0xFFFA:FB COP failure reset
    0xFFFC:FD Clock Monitor fail reset
*/
```

A.17 MC9S12DP256_FLAT.prm

```
NAMES
    msCANs12drv.o
END

SECTIONS
    REG_RG0 = NO_INIT 0x0000 TO 0x002F;
    PAGE_RG0= NO_INIT 0x0030 TO 0x0033;
    CRG_RG0 = NO_INIT 0x0034 TO 0x003F;
    TIM_RG0 = NO_INIT 0x0040 TO 0x007F;
    ATD_RG0 = NO_INIT 0x0080 TO 0x009F;
    PWM_RG0 = NO_INIT 0x00A0 TO 0x00C7;
    SCI_RG0 = NO_INIT 0x00C8 TO 0x00CF;
    SCI_RG1 = NO_INIT 0x00D0 TO 0x00D7;
    SPI_RG0 = NO_INIT 0x00D8 TO 0x00DF;
    IIC_RG0 = NO_INIT 0x00E0 TO 0x00E7;
    BDL_RG0 = NO_INIT 0x00E8 TO 0x00EF;
    SPI_RG1 = NO_INIT 0x00F0 TO 0x00F7;
    SPI_RG2 = NO_INIT 0x00F8 TO 0x00FF;

// from here on can not be short definitions
    FSH_RG0 = NO_INIT 0x0100 TO 0x010F;
    EE2_RG0 = NO_INIT 0x0110 TO 0x011B;
    ATD_RG1 = NO_INIT 0x0120 TO 0x013F;
    CAN_RG0 = NO_INIT 0x0140 TO 0x017F;
    CAN_RG1 = NO_INIT 0x0180 TO 0x01BF;
    CAN_RG2 = NO_INIT 0x01C0 TO 0x01FF;
    CAN_RG3 = NO_INIT 0x0200 TO 0x023F;
    PIM_RG0 = NO_INIT 0x0240 TO 0x026F;
    CAN_RG4 = NO_INIT 0x0280 TO 0x02BF;

    RAM = READ_WRITE 0x1010 TO 0x3FFF;
    /* unbanked FLASH ROM */
    ROM_4000 = READ_ONLY  0x4000 TO 0x7FFF;
    ROM_C000 = READ_ONLY  0xC000 TO 0xFEFF;
    EEPROM = READ_WRITE 0x0400 TO 0x0FFF;

    MSCAN0_START = NO_INIT  0x0140 TO 0x0140;
END

PLACEMENT
    _PRESTART, STARTUP,
    ROM_VAR, STRINGS,
    NON_BANKED, DEFAULT_ROM,
    COPY                  INTO  ROM_C000, ROM_4000;
    DEFAULT_RAM           INTO  RAM;

    CAN0_REG              INTO  CAN_RG0;
    CAN1_REG              INTO  CAN_RG1;
    CAN2_REG              INTO  CAN_RG2;
    CAN3_REG              INTO  CAN_RG3;
    CAN4_REG              INTO  CAN_RG4;
    FLSH_REG              INTO  FSH_RG0;
    EPRM_REG              INTO  EE2_RG0;
    BDLC_REG              INTO  BDL_RG0;
    IIC_REG               INTO  IIC_RG0;
    TIM_REG               INTO  TIM_RG0;
    PAGE_REG              INTO  PAGE_RG0;
    SCI0Regs              INTO  SCI_RG0;
    SCI1Regs              INTO  SCI_RG1;
    SPI0Regs              INTO  SPI_RG0;
    SPI1Regs              INTO  SPI_RG1;
    SPI2Regs              INTO  SPI_RG2;
    PWM_REG               INTO  PWM_RG0;
    REG_REG               INTO  REG_RG0;
    CRG_REG               INTO  CRG_RG0;
    ATD0_REG              INTO  ATD_RG0;
    ATD1_REG              INTO  ATD_RG1;
    PIM_REG               INTO  PIM_RG0;

    MSCAN0                INTO  MSCAN0_START;

END

STACKSIZE 0x100

VECTOR 0  _Startup
VECTOR ADDRESS 0xFFF0     rtiISR      /* real time interrupt service routine */
VECTOR ADDRESS 0xFFB0     CAN0_TransmitISR    /* CAN0 Tx */
VECTOR ADDRESS 0xFFB2     CAN0_ReceiveISR     /* CAN0 Rx */
VECTOR ADDRESS 0xFFB6     CAN0_WakeupISR      /* CAN0 Wake-up */


/*  Interrupt Vector Table */
/*  0xFF8C:8D PWM Emergency Shutdown
    0xFF8E:8F Port P Interrupt
    0xFF90:91 MSCAN 4 transmit
    0xFF92:93 MSCAN 4 receive
    0xFF94:95 MSCAN 4 errors
    0xFF96:97 MSCAN 4 wake- up
    0xFF98:99 MSCAN 3 transmit
    0xFF9A:9B MSCAN 3 receive
    0xFF9C:9D MSCAN 3 errors
    0xFF9E:9F MSCAN 3 wake- up
    0xFFA0:A1 MSCAN 2 transmit
    0xFFA2:A3 MSCAN 2 receive
    0xFFA4:A5 MSCAN 2 errors
    0xFFA6:A7 MSCAN 2 wake-up
    0xFFA8:A9 MSCAN 1 transmit
    0xFFAA:AB MSCAN 1 receive
    0xFFAC:AD MSCAN 1 errors
    0xFFAE:AF MSCAN 1 wake-up
    0xFFB0:B1 MSCAN 0 transmit
    0xFFB2:B3 MSCAN 0 receive
    0xFFB4:B5 MSCAN 0 errors
    0xFFB6:B7 MSCAN 0 wake-up
    0xFFB8:B9 FLASH
    0xFFBA:BB EEPROM
    0xFFBC:BD SPI2
    0xFFBE:BF SPI1
    0xFFC0:C1 IIC Bus
    0xFFC2:C3 DLC
    0xFFC4:C5 SCME
    0xFFC6:C7 CRG lock
    0xFFC8:C9 Pulse Accumulator B Overflow
    0xFFCA:CB Modulus Down Counter underflow
    0xFFCC:CD Port H
    0xFFCE:CF Port J
    0xFFD0:D1 ATD1
    0xFFD2:D3 ATD0
    0xFFD4:D5 SCI1
    0xFFD6:D7 SCI0
    0xFFD8:D9 SPI0
    0xFFDA:DB Pulse accumulator input edge
    0xFFDC:DD Pulse accumulator A overflow
    0xFFDE:DF Timer overflow
    0xFFE0:E1 Timer channel 7
    0xFFE2:E3 Timer channel 6
    0xFFE4:E5 Timer channel 5
    0xFFE6:E7 Timer channel 4
    0xFFE8:E9 Timer channel 3
    0xFFEA:EB Timer channel 2
    0xFFEC:ED Timer channel 1
    0xFFEE:EF Timer channel 0
    0xFFF0:F1 RTI - Real time interrupt
    0xFFF2:F3 IRQ
    0xFFF4:F5 XIRQ
    0xFFF6:F7 SWI
    0xFFF8:F9 Unimplemented instruction trap
    0xFFFA:FB COP failure reset
    0xFFFC:FD Clock Monitor fail reset
*/
```

Appendix B. Bill of Materials and Schematics

B.1 Contents

B.2 Industrial CAN I/O Module Bill of Materials

Table B-1. Base Board Bill of materials

CAN I/O_1 Revised: Wednesday, June 12, 2002
Revision: 0.5

MCSL Roznov
1. maje 1009
756 61 Roznov p.R., Czech Republic, Europe

Bill Of Mate Page1

Item	Quantity	Reference	Part		Supplier
1	71	C1,C2,C4,C5,C6,C7,C11, C12,C14,C15,C16,C17,C21, C22,C24,C25,C26,C27,C31, C32,C34,C35,C36,C37,C41, C42,C44,C45,C46,C47,C51, C52,C54,C55,C56,C57,C61, C62,C64,C65,C66,C67,C71, C72,C74,C75,C76,C77,C81, C82,C83,C84,C85,C86,C87, C88,C89,C90,C94,C96,C98, C100,C110,C115,C122,C124, C125,C126,C127,C128,C129	100nF	C0805	Farnell-499-687
2	14	C3,C13,C23,C33,C43,C53, C63,C73,C93,C95,C97,C99, C109,C114	10nF	C0805	Farnell-499-225
3	9	C8,C18,C28,C38,C48,C58, C68,C78,C130	1nF	C0805	Farnell-894-825
4	2	C92,C91	33pF	C0805	Farnell-317-603
5	3	C111,C116,C117	22uF/6.3V	AL-elyt	Farnell-556-117
6	1	C112	33nF	C0805	Farnell-894-886
7	1	C113	3.3 nF	C0805	Farnell-894-849
8	2	C119,C121	47nF	C0805	Farnell-894-898

9	1	C120	4.7nF	C0805	Farnell-894-850
10	8	D1,D4,D7,D10,D13,D16,D19, D22	BAV99LT1		OnSemiconductor
11	8	D2,D5,D8,D11,D14,D17,D20, D23	BAS40-04LT1		OnSemiconductor
12	1	D25	MBRS130LT3		OnSemiconductor
13	1	JP1	485FDX	Not Populated	
14	1	JP2	120DISC	Not Populated	
15	2	J1,J2	HEADER 10X2		Fischer elektronik-S
16	1	J3	CON18/MOLEX		Molex-861518
17	1	J4	HEADER 4X2		Fischer elektronik-S
18	1	J5	CON/5MOLEX		Farnell - .889-799
19	1	J6	CON/CANNON9/90DEG/FEMALE		
20	1	J7	CON/CANNON9/90DEG/MALE		
21	1	J8	HEADER 3X2		Fischer elektronik-S
22	3	L1,L2,L3	10uH	Inductor Axial	Farnell - 108-267
23	8	R1,R12,R23,R34,R45,R56, R67,R78	100k	R0805	Farnell - 912-098
24	8	R2,R13,R24,R35,R46,R57, R68,R79	1.5k	R0805	Farnell - 911-872
25	8	R3,R14,R25,R36,R47,R58, R69,R80	1k	R0805	Farnell - 911-859
26	40	R4,R7,R10,R11,R15,R18, R21,R22,R26,R29,R32,R33, R37,R40,R43,R44,R48,R51, R54,R55,R59,R62,R65,R66, R70,R73,R76,R77,R81,R84, R87,R88,R130,R131,R132, R133,R134,R135,R136,R137	10k/0,1%	R0805	Farnell - 554-960
27	8	R5,R16,R27,R38,R49,R60, R71,R82	3.3k	R0805	Farnell - 911-914
28	8	R6,R17,R28,R39,R50,R61, R72,R83	250R/0,5W0,01%	VISHAY-S102J	Farnell-309-8084
29	16	R8,R9,R19,R20,R30,R31, R41,R42,R52,R53,R63,R64, R74,R75,R85,R86	2.2k		Farnell-613-137

30	6	R114,R115,R117,R118,R122, R123	10k	R0805	Farnell - 911-975
31	1	R116	4.7k	R0805	Farnell - 911-938
32	1	R119	2.7K	R0805	Farnell - 911-902
33	2	R120,R124	510R	R0805	Farnell - 771-302
34	1	R121	33k	R0805	Farnell - 912-037
35	2	R128,R129	120R	R0805	Farnell - 911-744
36	8	SM1,SM2,SM3,SM4,SM5,SM6, SM7,SM8	Solder Meander	Not Populated	
37	1	SW1	Switch/DIP8	A6S8102	Farnell - 328-1917
38	9	U1,U4,U7,U10,U13,U16,U19, U22,U25	MC33078D	OnSemiconductor	
39	8	U2,U5,U8,U11,U14,U17,U20, U23	MAX4690EAE	Maxim	
40	1	U26	MC9S12DP256	Motorola	
41	1	U27	MC33388D	Motorola	
42	1	U28	MAX202ECSE	Maxim	
43	1	U29	MAX491ECSD	Maxim	
44	1	Y1	CRYSTAL-16.0MHz		Farnell - 639-588
45	1	Box - WEB-B9	GM Elektronic		GM/622-214

Table B-2. Power Supply Board Bill of materials

CAN I/O_1 Revised: Tuesday, June 11, 2002
Revision: 0.2

MCSL Roznov
1. maje 1009
756 61 Roznov p.R., Czech Republic, Europe

Bill Of Mate Page1

Item	Quantity	Reference	Part		Supplier
1	2	C1,C3	47uF/16V	AL Electrolyt	Farnell-556-180
4	4	C2,C4,C17,C19	1uF/50V	AL Electrolyt	Farnell-556-312
3	1	C5	100uF/25V	AL Electrolyt	Farnell-156-619
4	2	C6,C8	33uF/25V	AL Electrolyt	Farnell-556-221
5	1	C7	3.3uF/35V	AL Electrolyt	Farnell-556-233
6	5	C9,C10,C18,C20,C21	100nF	C0805	Farnell-499-687
7	1	C11	100nF/50V	C0805	Farnell-499-687
8	1	C12	1.5nF	C0805	Farnell-301-9883
9	1	C13	100pF	C0805	Farnell-894-746
10	1	C14	1nF	C0805	Farnell-894-825
11	2	C15,C22	10nF	C0805	Farnell-894-862
12	1	C16	47uF/6.3V	AL Electrolyt	Farnell-556-129
13	3	D1,D2,D3	MBRS130LT3		Onsemiconductor
14	1	D4	MBRS340T3		Onsemiconductor
15	3	GC1,GC2,GC3	Ground_Connection		NOT populated
16	1	J1	HEADER 2x10	BL2/20Z	Fischer Elektronik
17	1	J2	MALE HEADER 2x10	SL 11 SMD 104/20	Fischer Elektronik
18	1	J3	HEADER 2x10	BL2/20Z	Fischer Elektronik
19	1	J4	HEADER 2x4	BL2/8Z	Fischer Elektronik
20	2	L2,L1	4.7uH		RS Components-367
21	1	PC1	TEN 4-2422		NOT populated
22	1	R1	680R	R0805	Farnell-613-071

23	1	R2	30k	R0805	Farnell-771-510
24	2	R5,R3	22k	R0805	Farnell-613-253
25	1	R4	100k	R0805	Farnell-613-332
26	1	R6	47k	R0805	Farnell-613-290
27	1	R7	33R/2W	UR001	
28	1	R8	20k	R0805	Farnell-771-491
29	2	SM1,SM2	Solder Meander		NOT populated
30	1	TP1	VPRE		NOT populated
31	1	TP2	VDD		NOT populated
32	1	TP3	A5V		NOT populated
33	1	TP4	R5V		NOT populated
34	1	TP5	Testpiont		NOT populated
35	1	TP6	+12VA		NOT populated
36	1	TP7	-12VA		NOT populated
37	1	T1	Tr_Q4437_B		Coilcraft/Q4437-B
38	1	U1	MC78M12BDT		Onsemiconductor
39	1	U2	MC79M12BDT		Onsemiconductor
40	1	U3	MC33394DH		Motorola

Table B-3. I/O Board Bill of material

CAN I/O_1 Revised: Monday, June 10, 2002
Revision: 0.1

MCSL Roznov
1. maje 1009
756 61 Roznov p.R., Czech Republic, Europe

Bill O Page1

Item	Quantity	Reference	Part		Supplier
1	8	C1,C2,C3,C4,C19,C20,C21, C22	100nF	C0805	Farnell-499-687
2	16	C5,C6,C7,C8,C9,C10,C11, C12,C13,C14,C15,C16,C17, C18,C23,C24	10nF/200V	C1206	Farnell-491-202
3	16	D1,D2,D3,D4,D5,D6,D7,D8, D9,D10,D11,D12,D13,D14, D15,D16	GREEN LED	KA3528LSGT	
4	1	J1	HEADER 2x10	BL2/20Z	Fischer Elektronik
5	2	J9,J8	CON18/MOLEX	M861518	MOLEX
6	16	R1,R2,R3,R4,R5,R6,R7,R8, R9,R10,R11,R12,R13,R14, R15,R16	22k	R0805	Farnell-613-253
7	2	R18,R17	130k	R0805	Farnell-771-594
8	2	U2,U3	MC33884DW		Motorola
9	2	U7,U8	MC33298DW		Motorola

B.3 Industrial CAN I/O Module Schematics

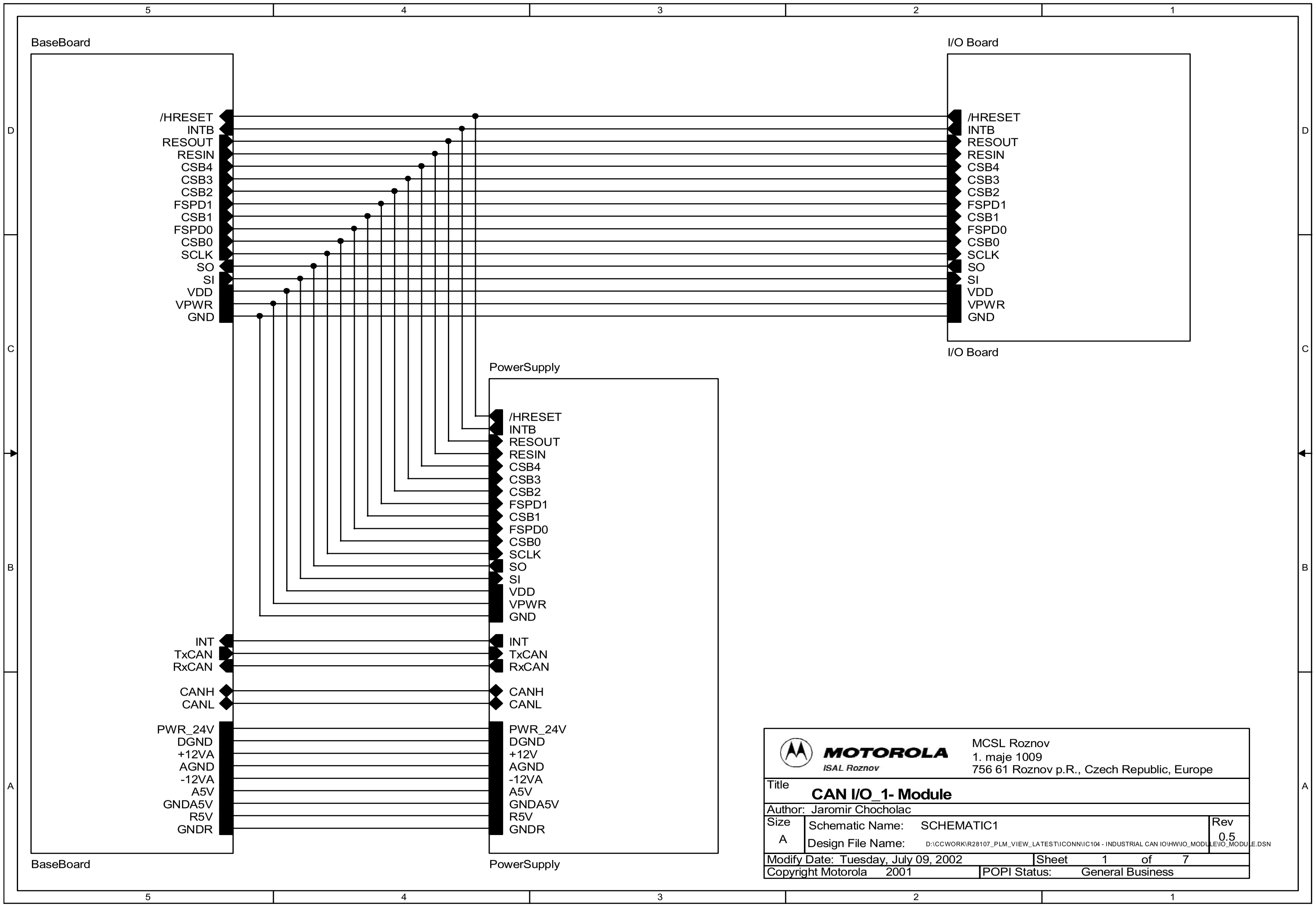


Figure B-1. I/O MODULE BLOCKS

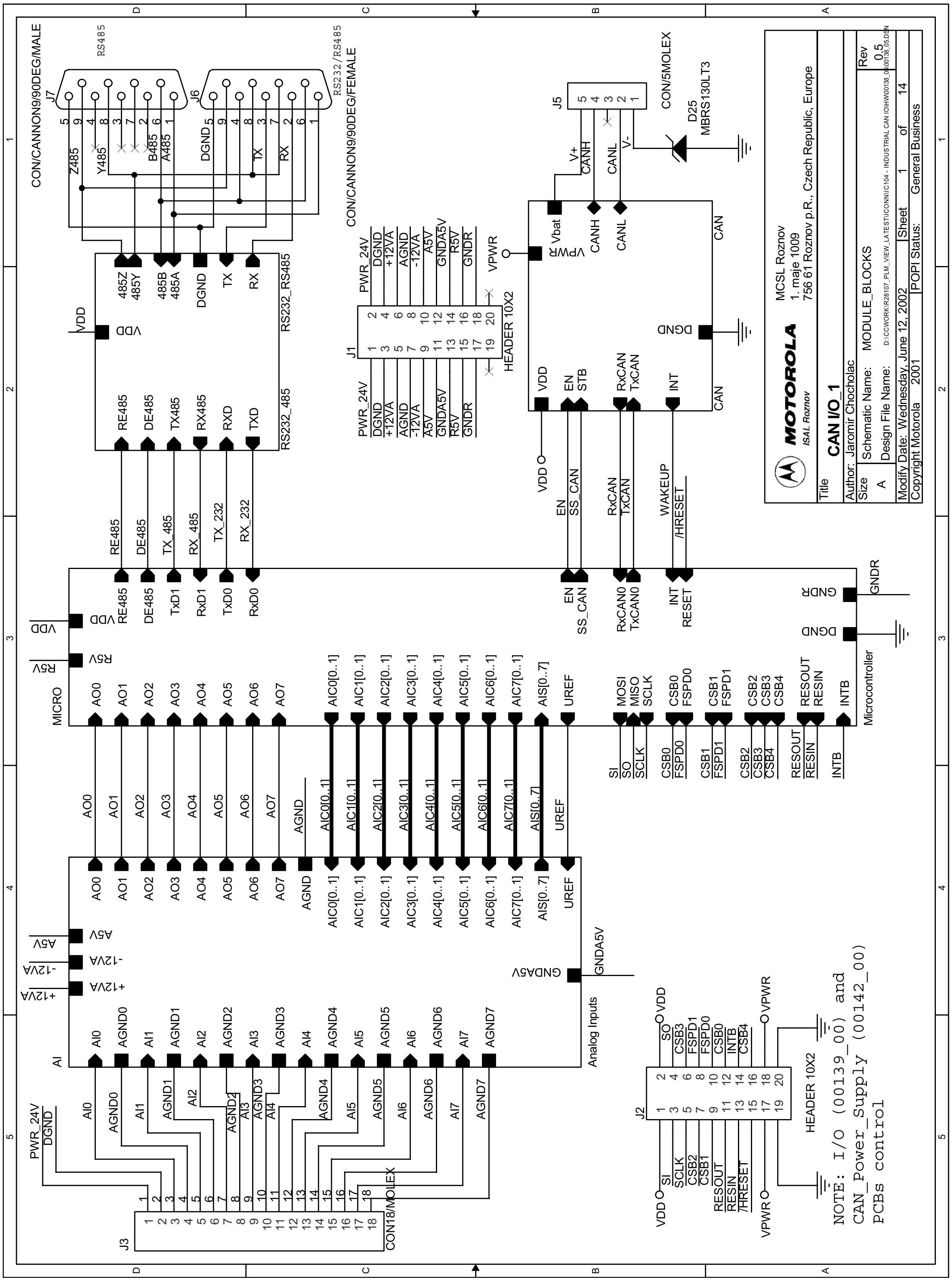


Figure B-2. BASE BOARD BLOCKS

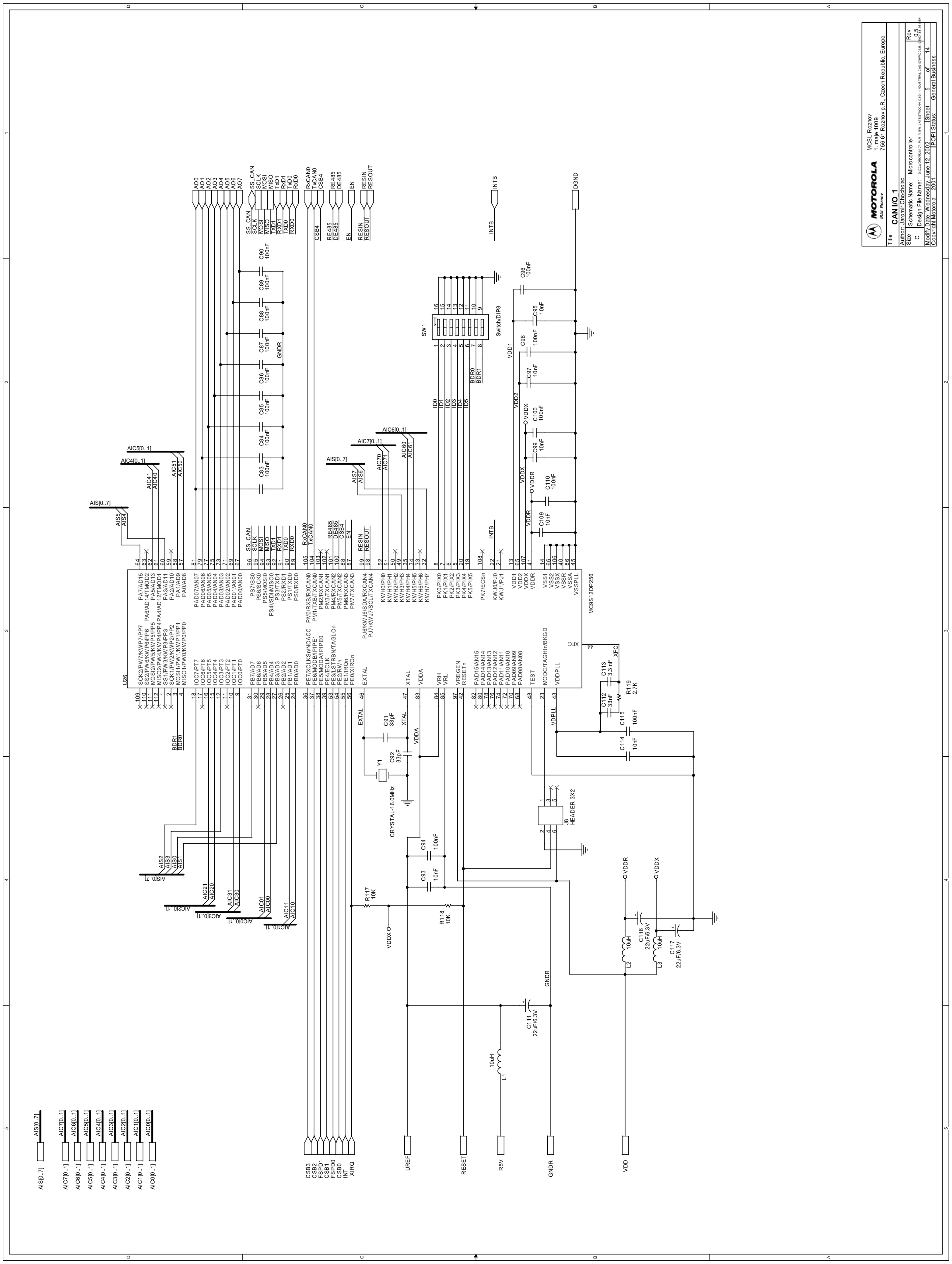


Figure B-3. MICROCONTROLLER

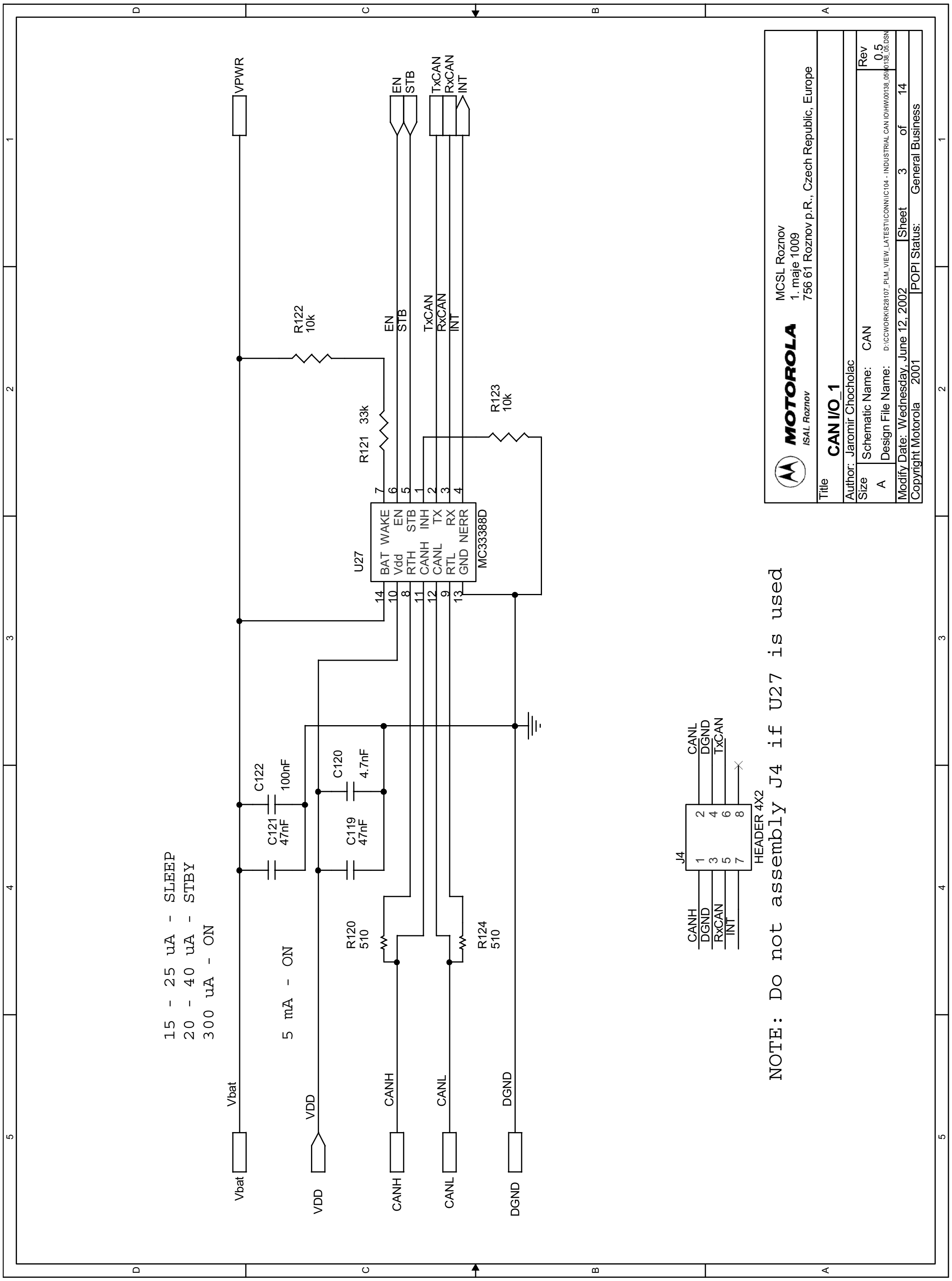


Figure B-4. CAN

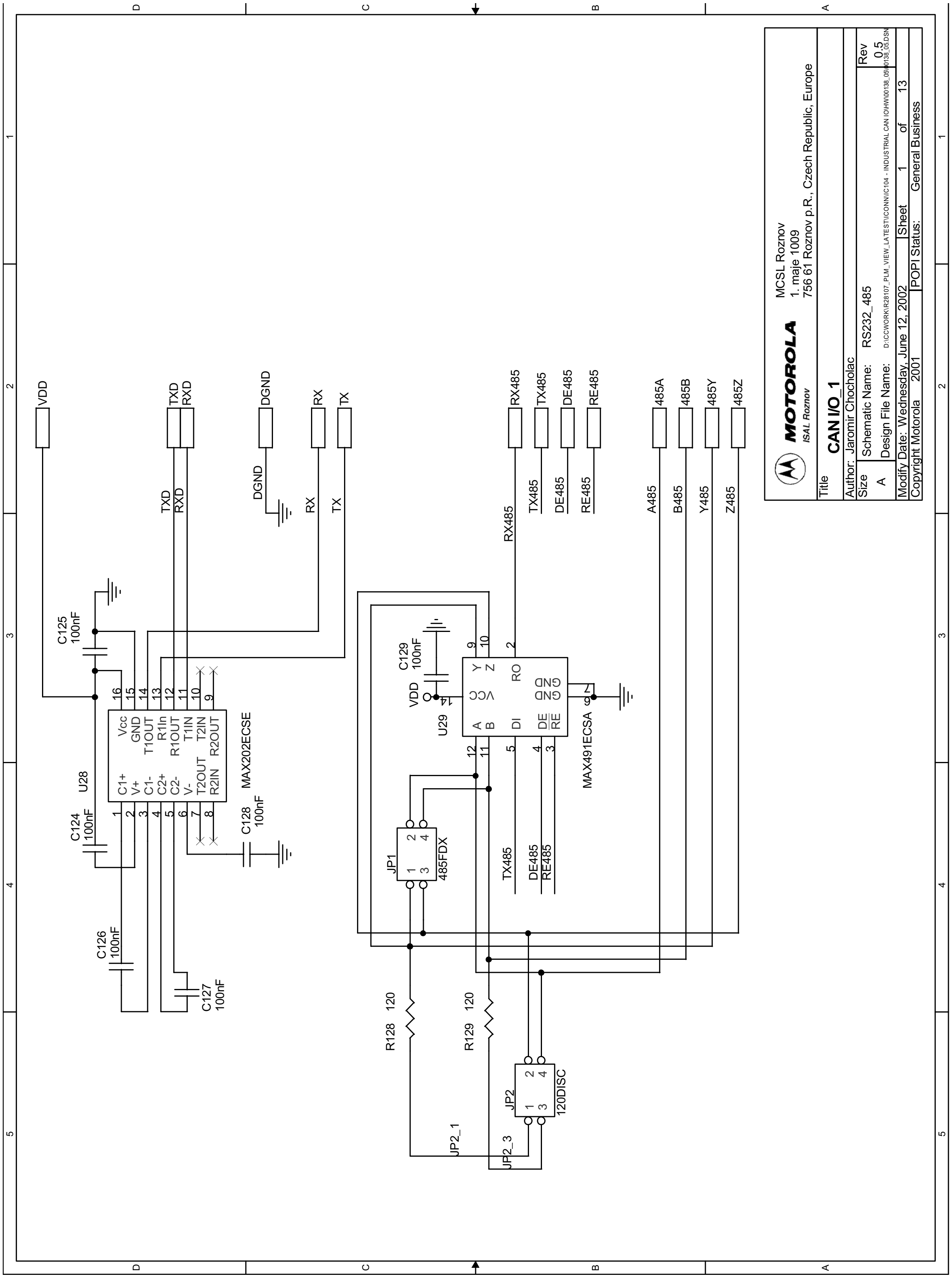


Figure B-5. RS232_485

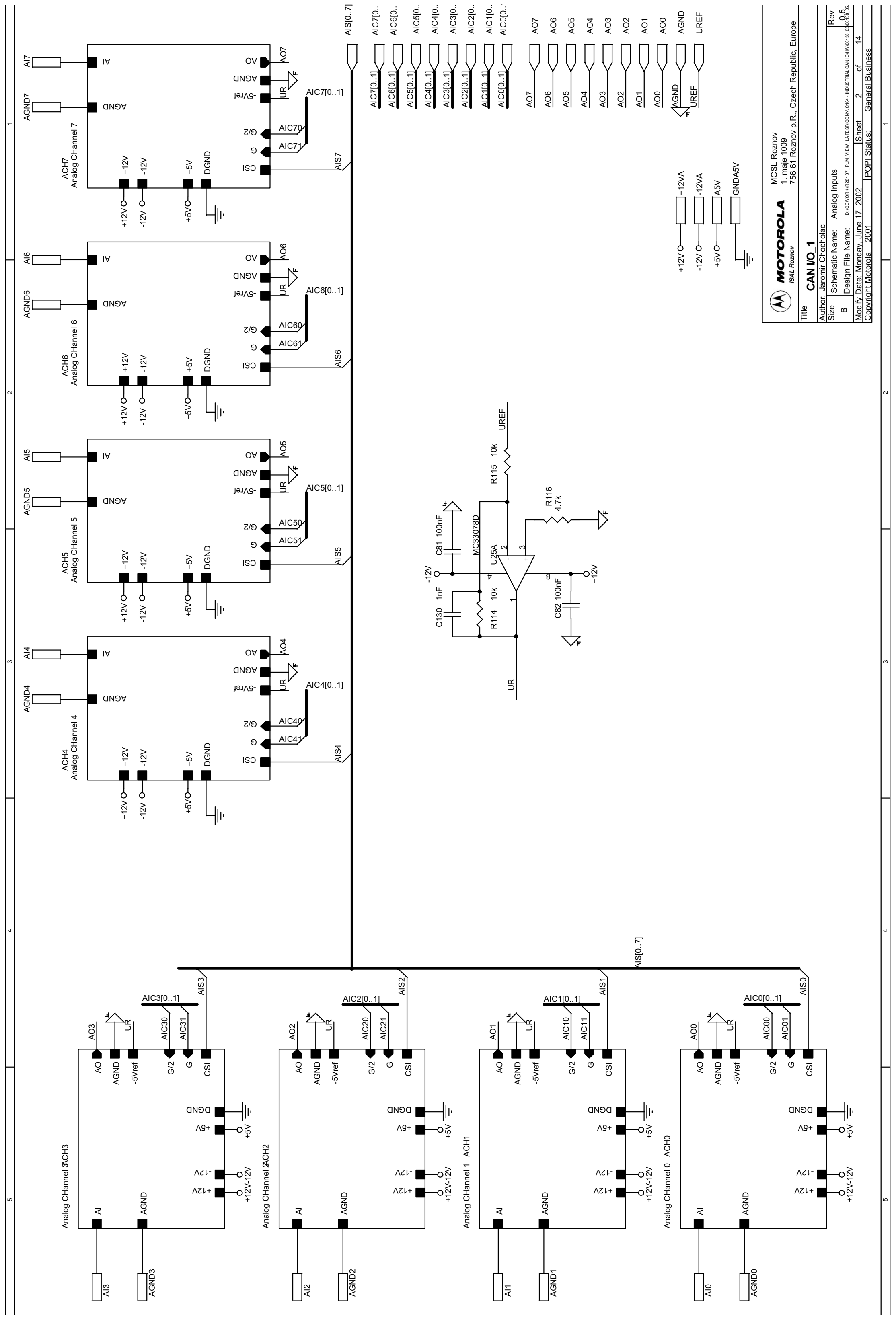


Figure B-6. ANALOG INPUTS

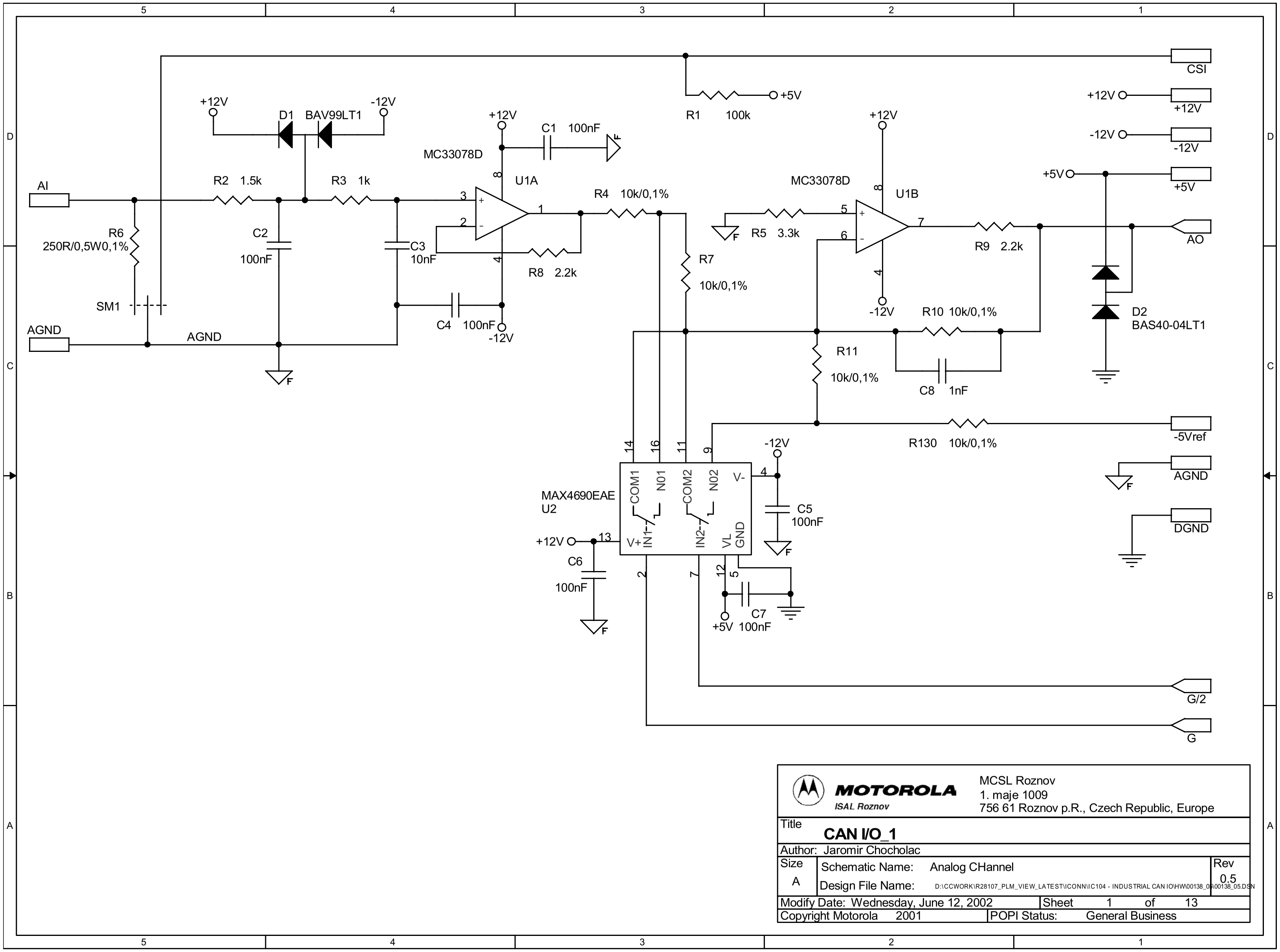


Figure B-7. ANALOG CHANNEL #0

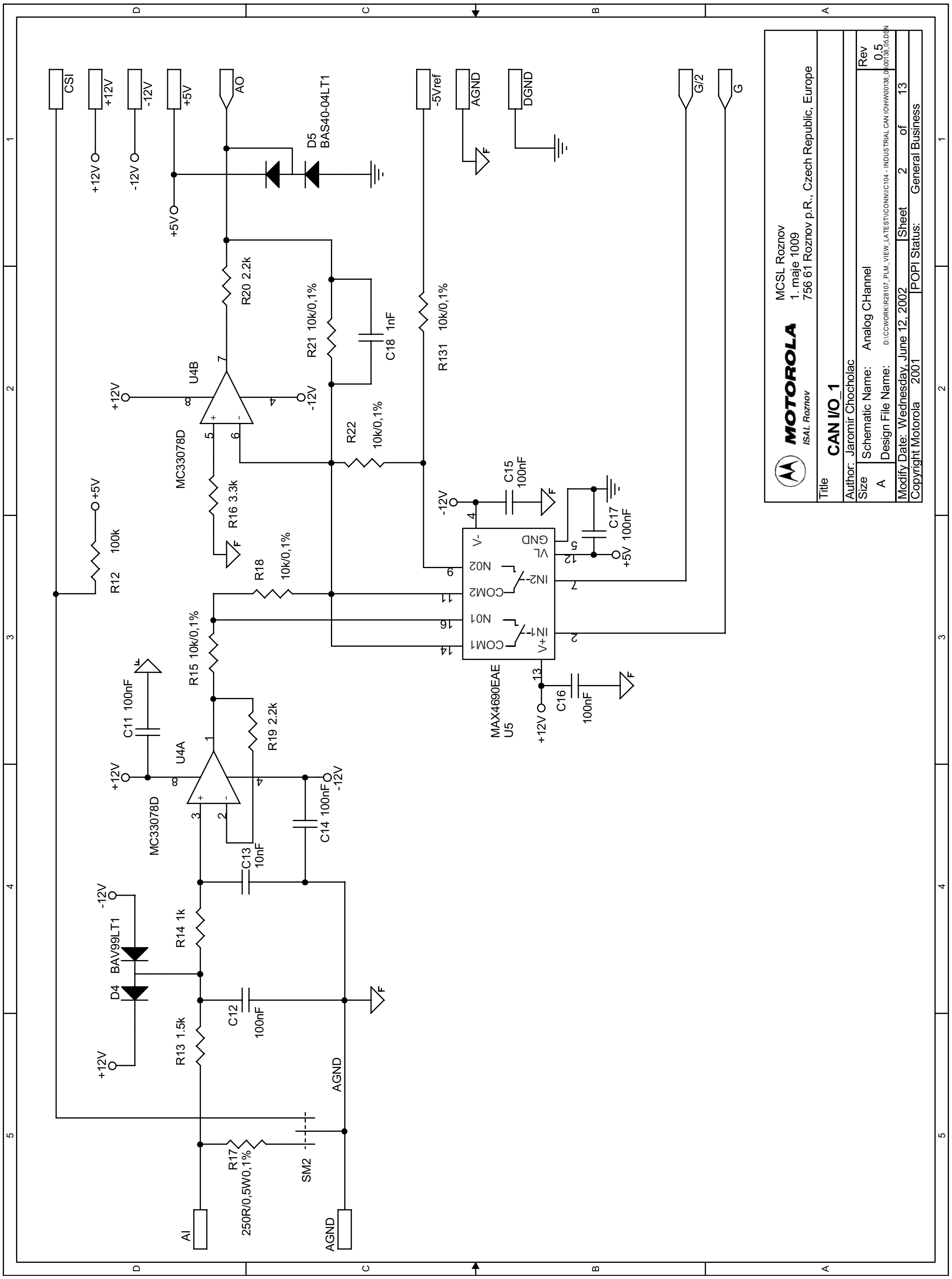


Figure B-8. ANALOG CHANNEL #1

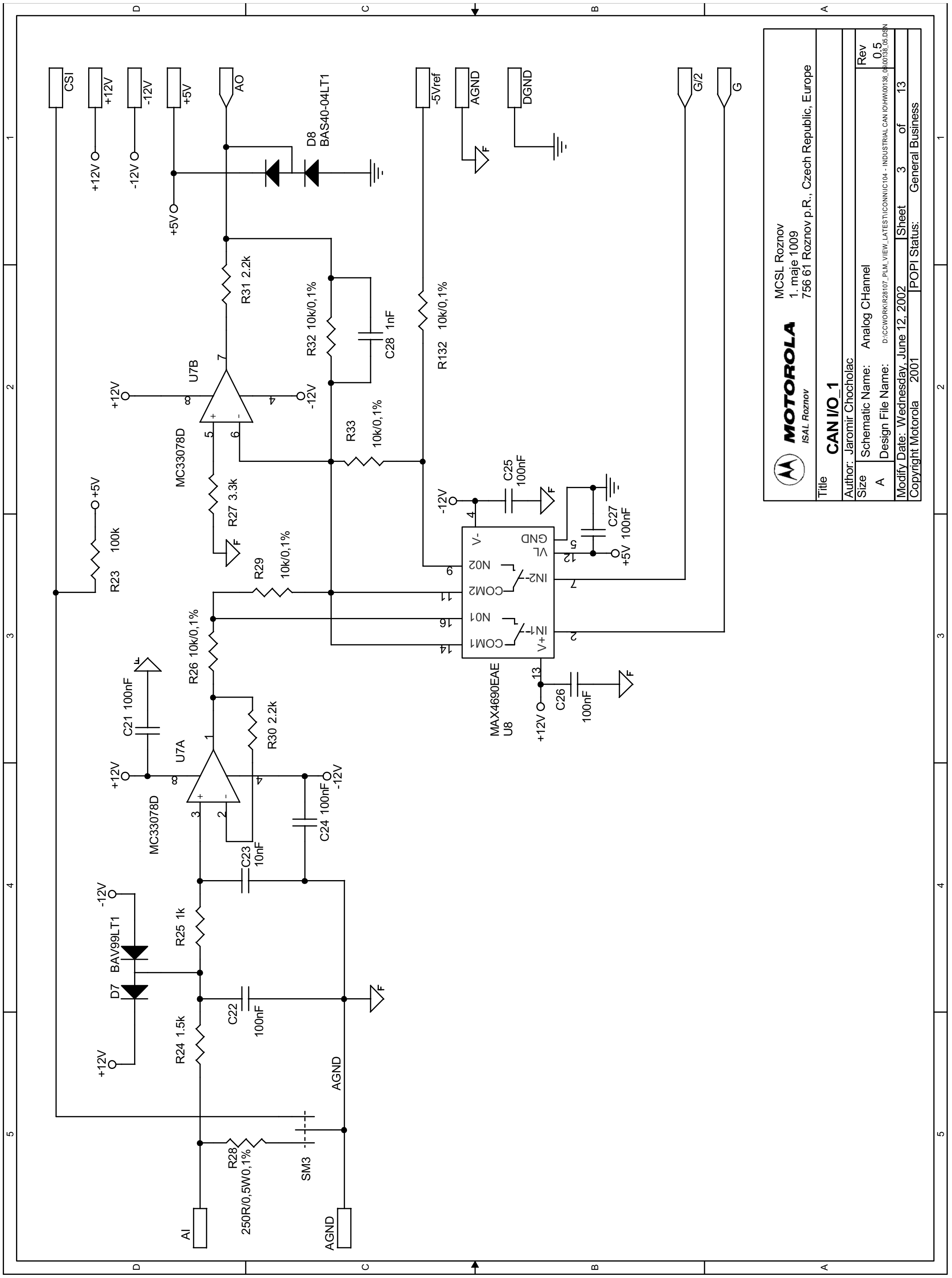


Figure B-9. ANALOG CHANNEL #2

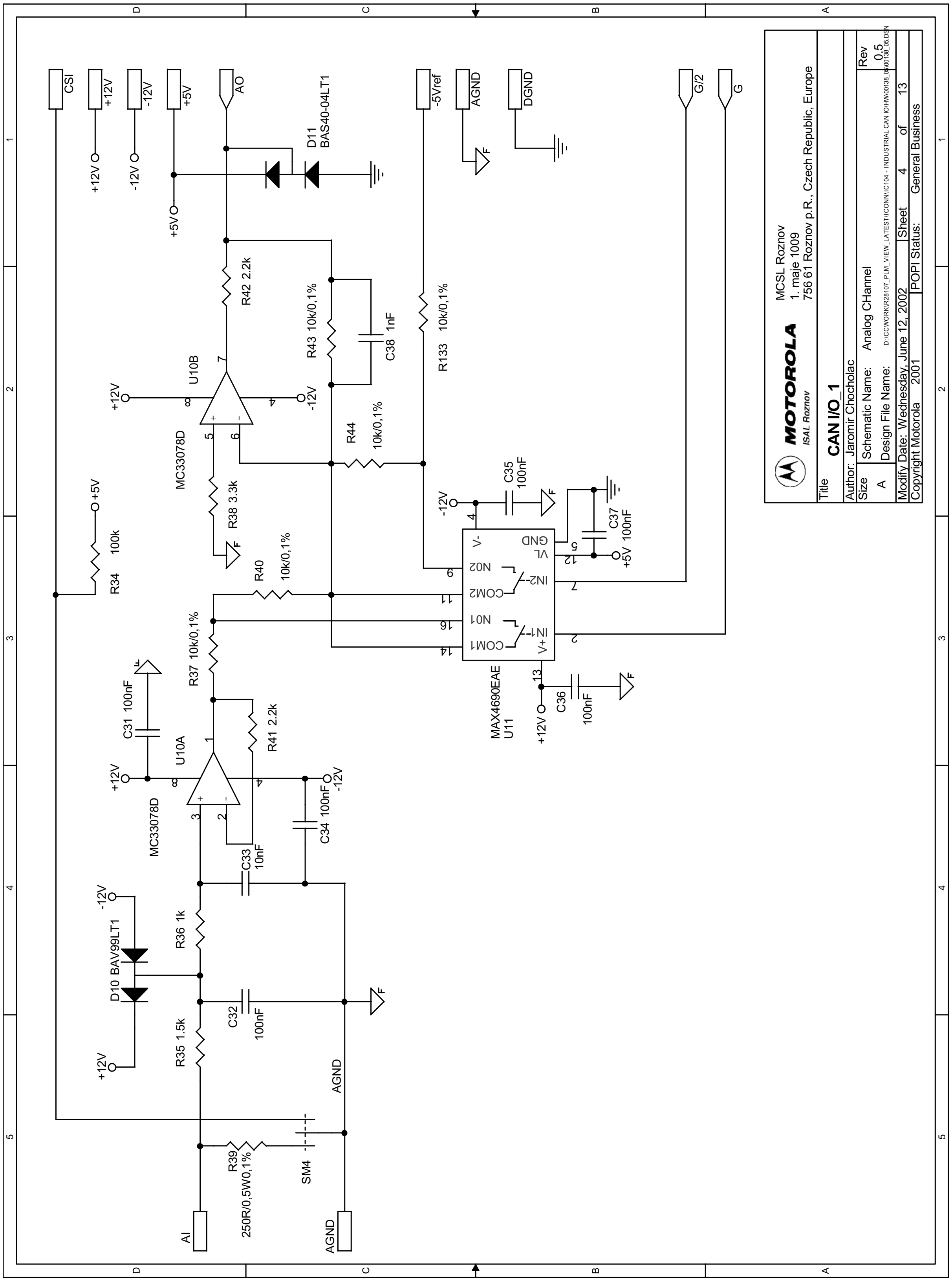


Figure B-10. ANALOG CHANNEL #3

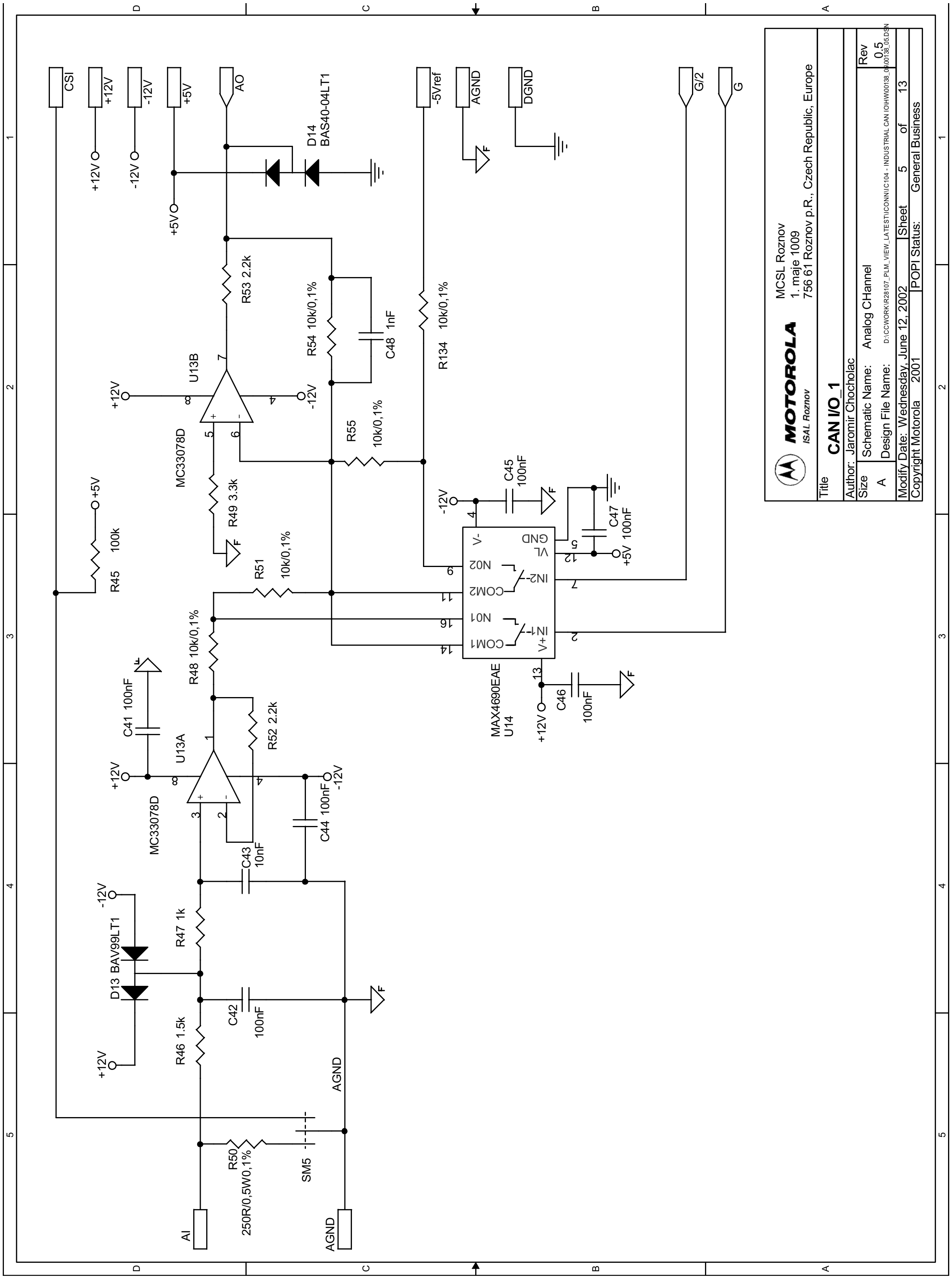


Figure B-11. ANALOG CHANNEL #4

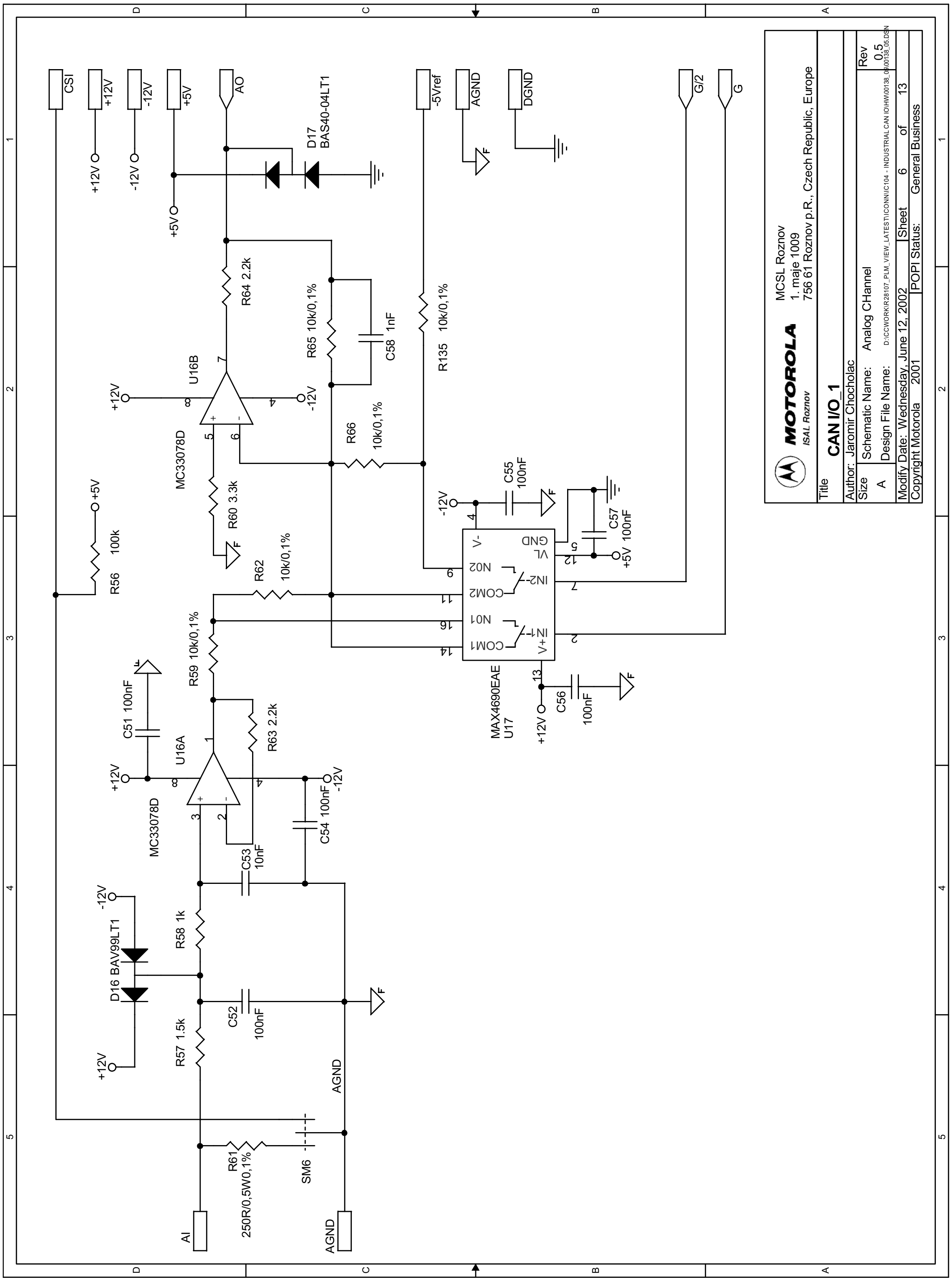


Figure B-12. ANALOG CHANNEL #5

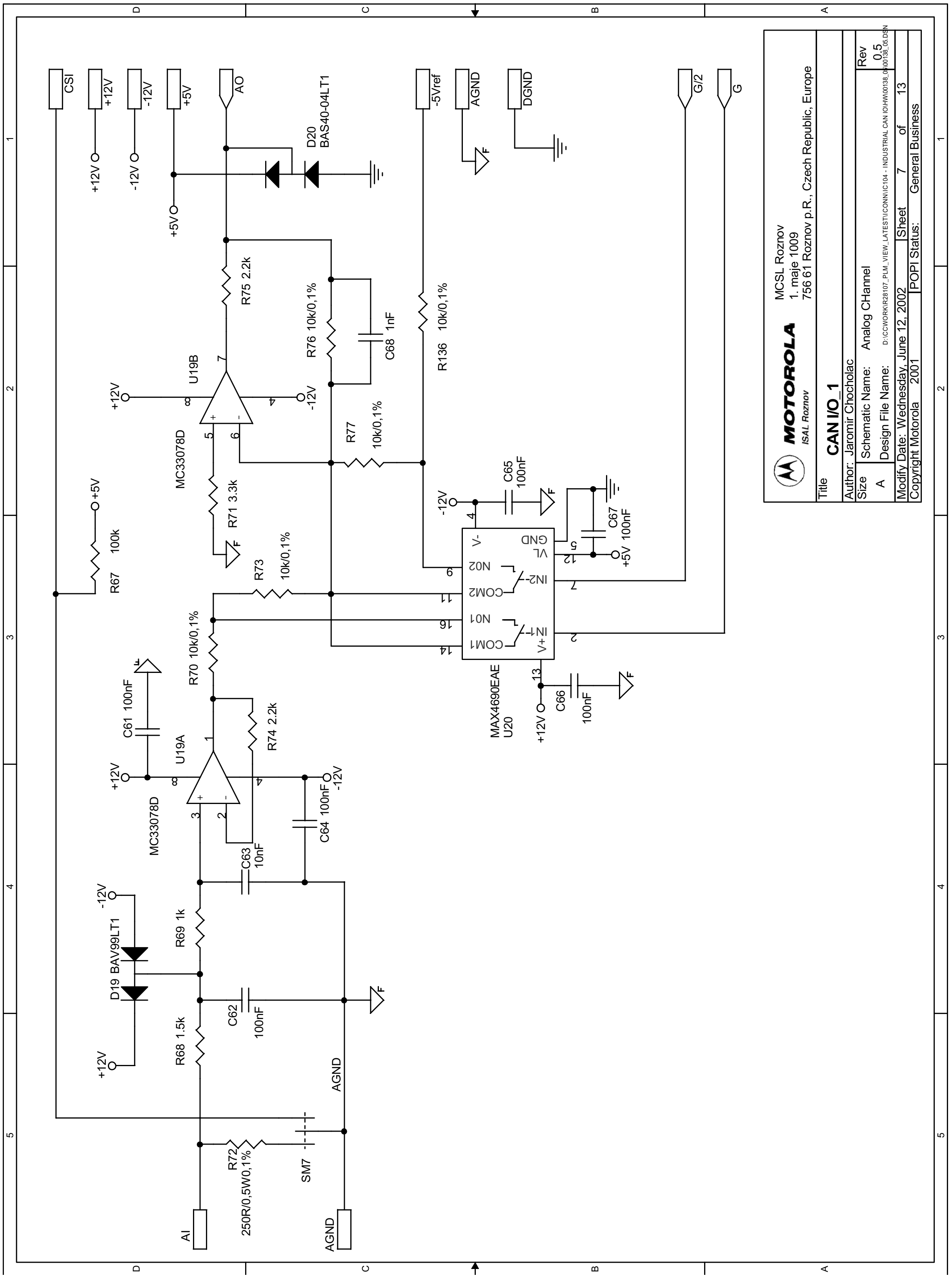


Figure B-13. ANALOG CHANNEL #6

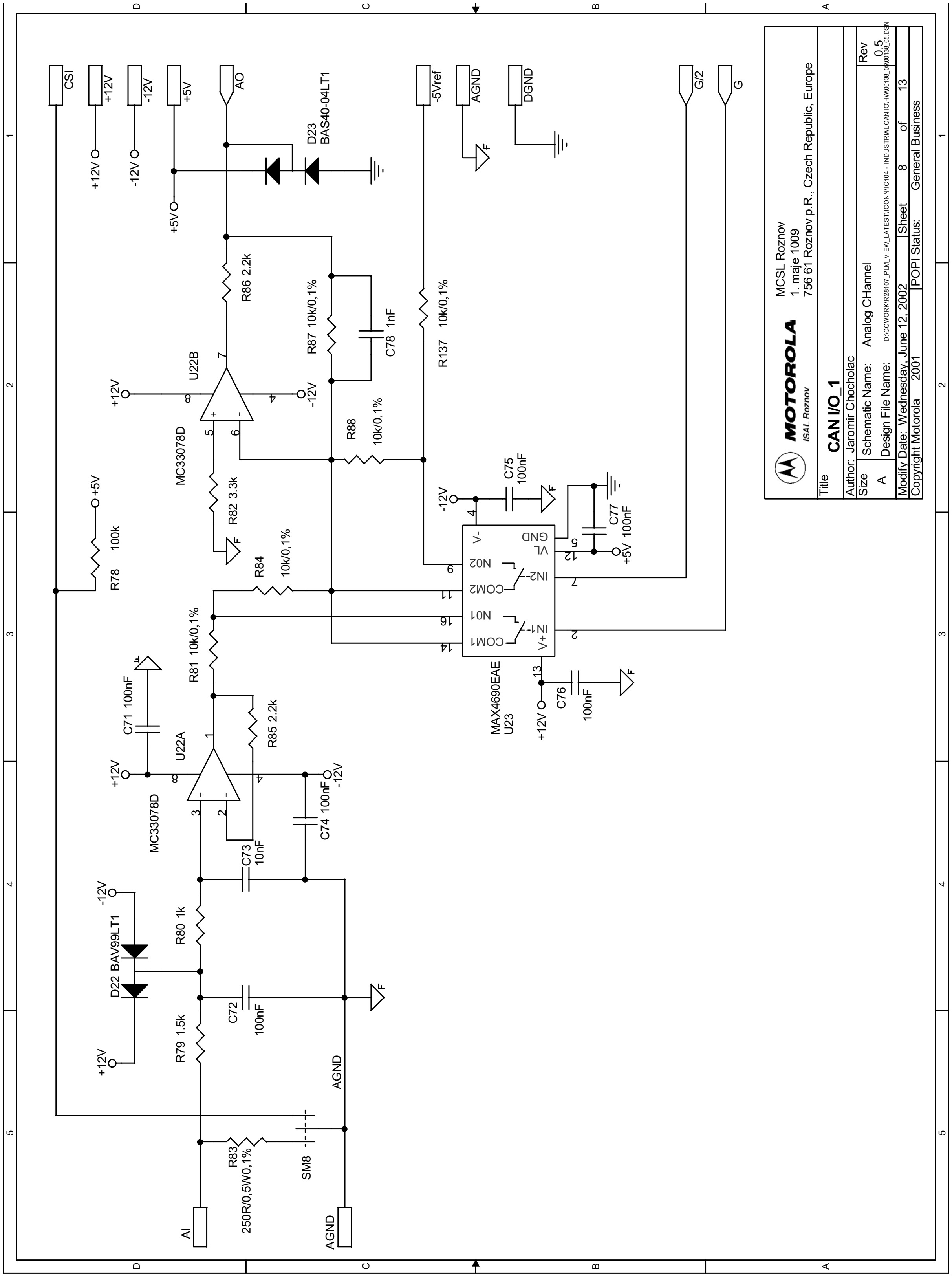


Figure B-14. ANALOG CHANNEL #7

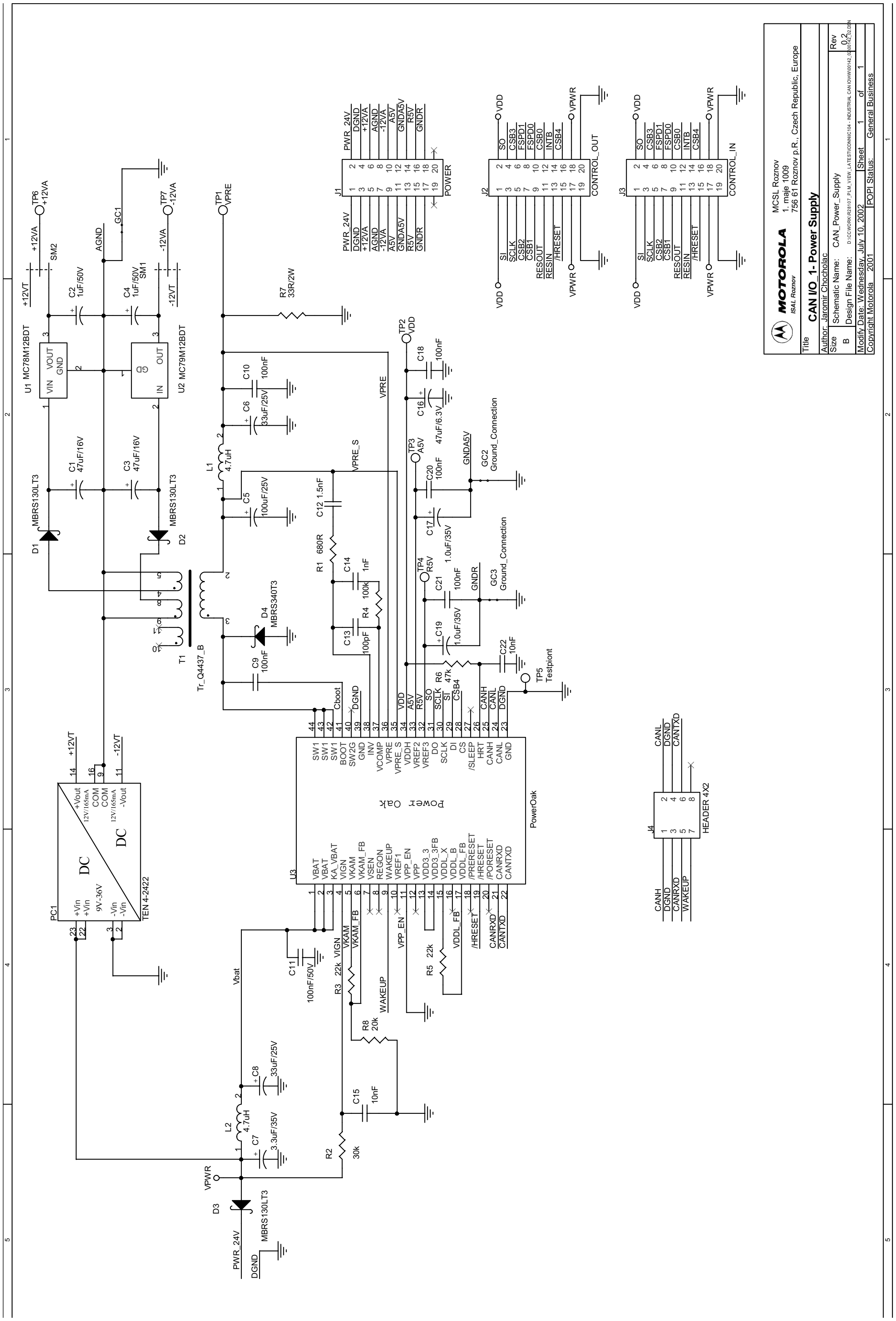


Figure B-15. POWER SUPPLY

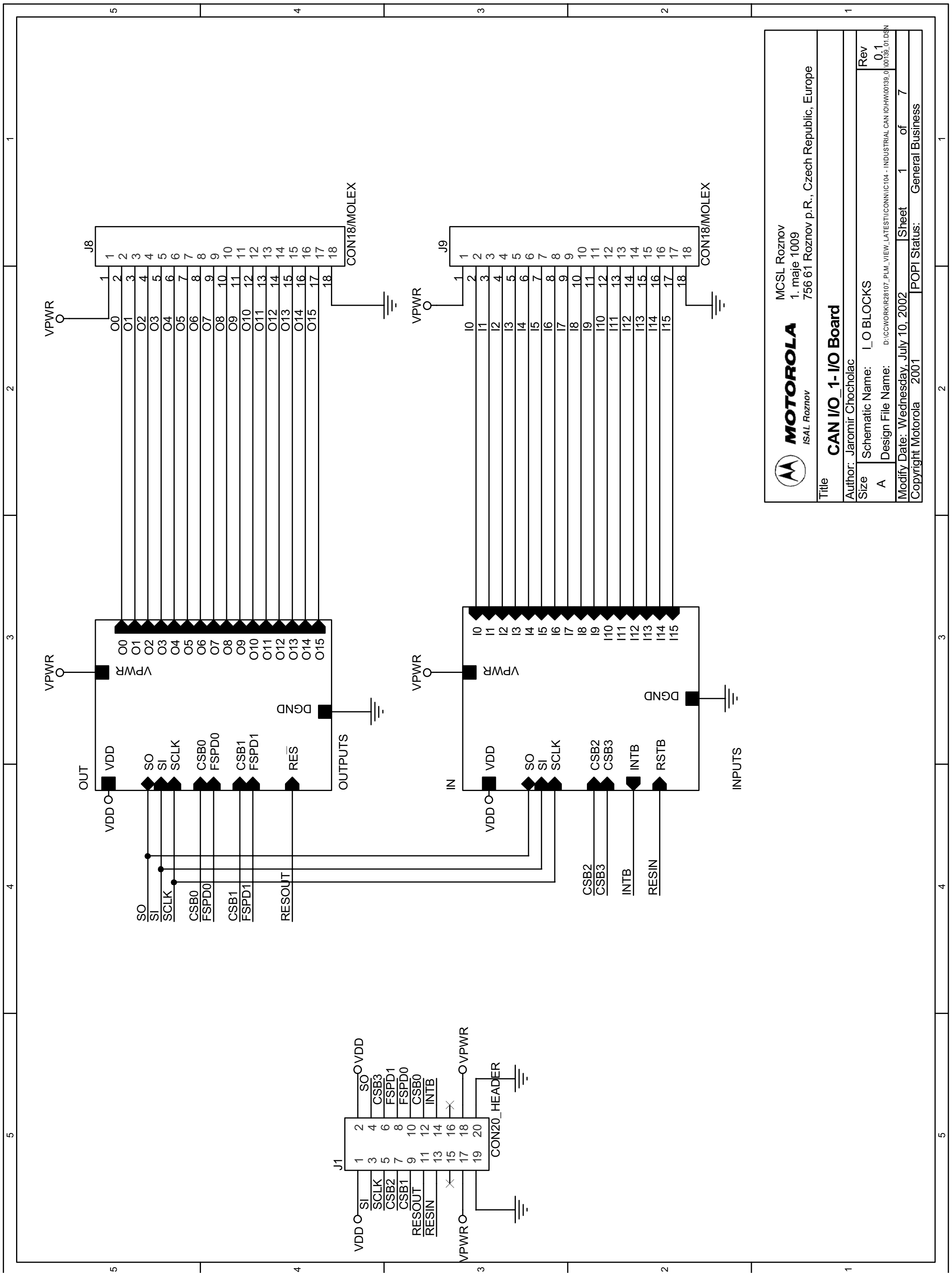

VPWR
J8
CON18/MOLEX
O0 O1 O2 O3 O4 O5 O6 O7 O8 O9 O10 O11 O12 O13 O14 O15
J9
I0 I1 I2 I3 I4 I5 I6 I7 I8 I9 I10 I11 I12 I13 I14 I15
OUT
VDD
SO
SI
SCLK
CSB0
FSPD0
CSB1
FSPD1
RES
OUTPUTS
DGND
IN
CSB2
CSB3
INTB
RSTB
INPUTS
RESOUT
RESIN
J1
CON20_HEADER
MOTOROLA
ISAL Roznov
MCSL Roznov
1. maje 1009
756 61 Roznov p.R., Czech Republic, Europe
Title
CAN I/O_1- I/O Board
Author: Jaromir Chocholac
Size
A
Schematic Name: I_O BLOCKS
Design File Name:
Rev
0.1
Modify Date: Wednesday, July 10, 2002
Sheet 1 of 7
Copyright Motorola 2001
POPI Status: General Business

Figure B-16. I/O BLOCKS

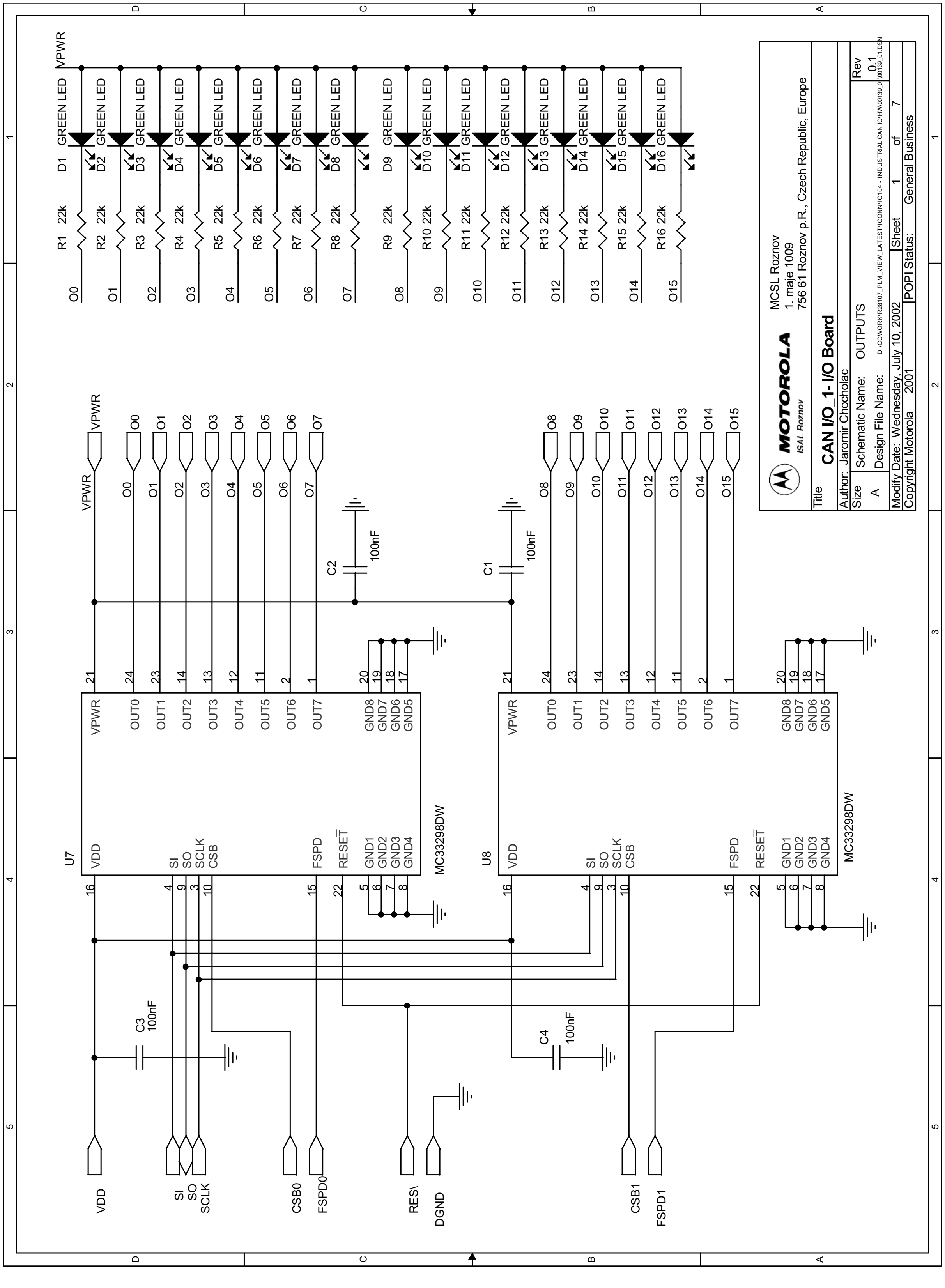


Figure B-17. OUTPUTS

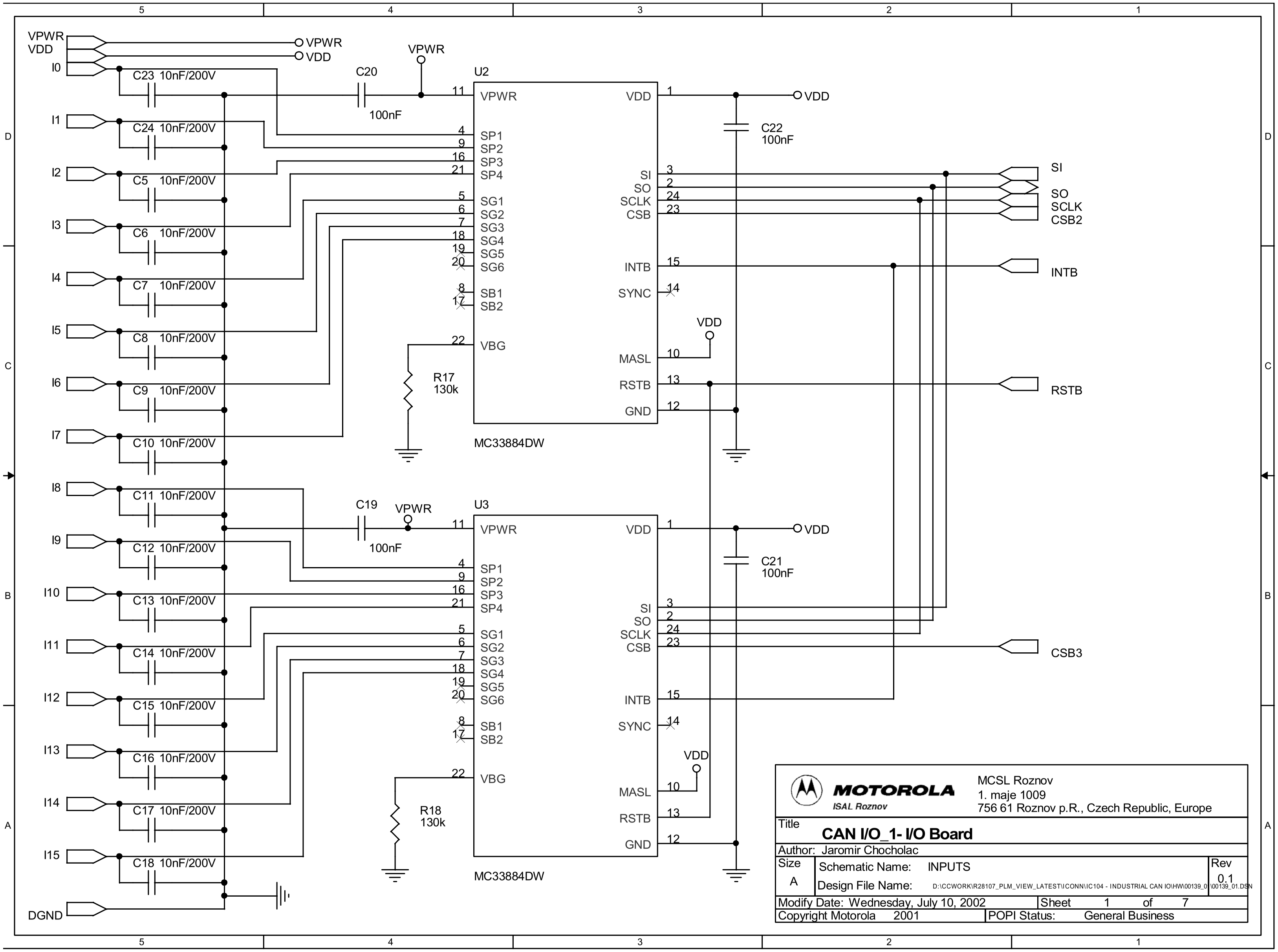


Figure B-18. INPUTS

Appendix C. Glossary

A — See “accumulators (A and B or D).”

accumulators (A and B or D) — Two 8-bit (A and B) or one 16-bit (D) general-purpose registers in the CPU. The CPU uses the accumulators to hold operands and results of arithmetic and logic operations.

acquisition mode — A mode of PLL operation with large loop bandwidth. Also see ’tracking mode’.

address bus — The set of wires that the CPU or DMA uses to read and write memory locations.

addressing mode — The way that the CPU determines the operand address for an instruction. The M68HC12 CPU has 15 addressing modes.

ALU — See “arithmetic logic unit (ALU).”

analogue-to-digital converter (ATD) — The ATD module is an 8-channel, multiplexed-input successive-approximation analog-to-digital converter.

arithmetic logic unit (ALU) — The portion of the CPU that contains the logic circuitry to perform arithmetic, logic, and manipulation operations on operands.

asynchronous — Refers to logic circuits and operations that are not synchronized by a common reference signal.

ATD — See “analogue-to-digital converter”.

B — See “accumulators (A and B or D).”

baud rate — The total number of bits transmitted per unit of time.

BCD — See “binary-coded decimal (BCD).”

binary — Relating to the base 2 number system.

binary number system — The base 2 number system, having two digits, 0 and 1. Binary arithmetic is convenient in digital circuit design because digital circuits have two permissible voltage levels, low and high. The binary digits 0 and 1 can be interpreted to correspond to the two digital voltage levels.

binary-coded decimal (BCD) — A notation that uses 4-bit binary numbers to represent the 10 decimal digits and that retains the same positional structure of a decimal number. For example,

234 (decimal) = 0010 0011 0100 (BCD)

bit — A binary digit. A bit has a value of either logic 0 or logic 1.

branch instruction — An instruction that causes the CPU to continue processing at a memory location other than the next sequential address.

break module — The break module allows software to halt program execution at a programmable point in order to enter a background routine.

breakpoint — A number written into the break address registers of the break module. When a number appears on the internal address bus that is the same as the number in the break address registers, the CPU executes the software interrupt instruction (SWI).

break interrupt — A software interrupt caused by the appearance on the internal address bus of the same value that is written in the break address registers.

bus — A set of wires that transfers logic signals.

bus clock — See "CPU clock".

byte — A set of eight bits.

CAN — See "Motorola scalable CAN."

CCR — See “condition code register.”

central processor unit (CPU) — The primary functioning unit of any computer system. The CPU controls the execution of instructions.

CGM — See “clock generator module (CGM).”

clear — To change a bit from logic 1 to logic 0; the opposite of set.

clock — A square wave signal used to synchronize events in a computer.

clock generator module (CGM) — The CGM module generates a base clock signal from which the system clocks are derived. The CGM may include a crystal oscillator circuit and/or phase-locked loop (PLL) circuit.

comparator — A device that compares the magnitude of two inputs. A digital comparator defines the equality or relative differences between two binary numbers.

computer operating properly module (COP) — A counter module that resets the MCU if allowed to overflow.

condition code register (CCR) — An 8-bit register in the CPU that contains the interrupt mask bit and five bits that indicate the results of the instruction just executed.

control bit — One bit of a register manipulated by software to control the operation of the module.

control unit — One of two major units of the CPU. The control unit contains logic functions that synchronize the machine and direct various operations. The control unit decodes instructions and generates the internal control signals that perform the requested operations. The outputs of the control unit drive the execution unit, which contains the arithmetic logic unit (ALU), CPU registers, and bus interface.

COP — See "computer operating properly module (COP)."

CPU — See “central processor unit (CPU).”

CPU12 — The CPU of the MC68HC12 Family.

CPU clock — Bus clock select bits BCSP and BCSS in the clock select register (CLKSEL) determine which clock drives SYSCLK for the main system, including the CPU and buses. When EXTALi drives the SYSCLK, the CPU or bus clock frequency (f_o) is equal to the EXTALi frequency divided by 2.

CPU cycles — A CPU cycle is one period of the internal bus clock, normally derived by dividing a crystal oscillator source by two or more so the high and low times will be equal. The length of time required to execute an instruction is measured in CPU clock cycles.

CPU registers — Memory locations that are wired directly into the CPU logic instead of being part of the addressable memory map. The CPU always has direct access to the information in these registers. The CPU registers in an M68HC12 are:

- A (8-bit accumulator)
- B (8-bit accumulator)
  - D (16-bit accumulator formed by concatenation of accumulators A and B)
- IX (16-bit index register)
- IY (16-bit index register)

- SP (16-bit stack pointer)
- PC (16-bit program counter)
- CCR (8-bit condition code register)

cycle time — The period of the operating frequency: $t_{CYC} = 1/f_{OP}$.

D — See “accumulators (A and B or D).”

decimal number system — Base 10 numbering system that uses the digits zero through nine.

duty cycle — A ratio of the amount of time the signal is on versus the time it is off. Duty cycle is usually represented by a percentage.

ECT — See “enhanced capture timer.”

EEPROM — Electrically erasable, programmable, read-only memory. A nonvolatile type of memory that can be electrically erased and reprogrammed.

EPROM — Erasable, programmable, read-only memory. A nonvolatile type of memory that can be erased by exposure to an ultraviolet light source and then reprogrammed.

enhanced capture timer (ECT) — The HC12 Enhanced Capture Timer module has the features of the HC12 Standard Timer module enhanced by additional features in order to enlarge the field of applications.

exception — An event such as an interrupt or a reset that stops the sequential execution of the instructions in the main program.

fetch — To copy data from a memory location into the accumulator.

firmware — Instructions and data programmed into nonvolatile memory.

free-running counter — A device that counts from zero to a predetermined number, then rolls over to zero and begins counting again.

full-duplex transmission — Communication on a channel in which data can be sent and received simultaneously.

hexadecimal — Base 16 numbering system that uses the digits 0 through 9 and the letters A through F.

high byte — The most significant eight bits of a word.

illegal address — An address not within the memory map

illegal opcode — A nonexistent opcode.

index registers (IX and IY) — Two 16-bit registers in the CPU. In the indexed addressing modes, the CPU uses the contents of IX or IY to determine the effective address of the operand. IX and IY can also serve as a temporary data storage locations.

input/output (I/O) — Input/output interfaces between a computer system and the external world. A CPU reads an input to sense the level of an external signal and writes to an output to change the level on an external signal.

instructions — Operations that a CPU can perform. Instructions are expressed by programmers as assembly language mnemonics. A CPU interprets an opcode and its associated operand(s) and instruction.

inter-IC bus (I^2C) — A two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange between devices.

interrupt — A temporary break in the sequential execution of a program to respond to signals from peripheral devices by executing a subroutine.

interrupt request — A signal from a peripheral to the CPU intended to cause the CPU to execute a subroutine.

I/O — See “input/output (I/0).”

jitter — Short-term signal instability.

latch — A circuit that retains the voltage level (logic 1 or logic 0) written to it for as long as power is applied to the circuit.

latency — The time lag between instruction completion and data movement.

least significant bit (LSB) — The rightmost digit of a binary number.

logic 1 — A voltage level approximately equal to the input power voltage (V_{DD}).

logic 0 — A voltage level approximately equal to the ground voltage (V_{SS}).

low byte — The least significant eight bits of a word.

M68HC12 — A Motorola family of 16-bit MCUs.

mark/space — The logic 1/logic 0 convention used in formatting data in serial communication.

mask — 1. A logic circuit that forces a bit or group of bits to a desired state. 2. A photomask used in integrated circuit fabrication to transfer an image onto silicon.

MCU — Microcontroller unit. See “microcontroller.”

memory location — Each M68HC12 memory location holds one byte of data and has a unique address. To store information in a memory location, the CPU places the address of the location on the address bus, the data information on the data bus, and asserts the write signal. To read information from a memory location, the CPU places the address of the location on the address bus and asserts the read signal. In response to the read signal, the selected memory location places its data onto the data bus.

memory map — A pictorial representation of all memory locations in a computer system.

MI-Bus — See "Motorola interconnect bus".

microcontroller — Microcontroller unit (MCU). A complete computer system, including a CPU, memory, a clock oscillator, and input/output (I/O) on a single integrated circuit.

modulo counter — A counter that can be programmed to count to any number from zero to its maximum possible modulus.

most significant bit (MSB) — The leftmost digit of a binary number.

Motorola interconnect bus (MI-Bus) — The Motorola Interconnect Bus (MI Bus) is a serial communications protocol which supports distributed real-time control efficiently and with a high degree of noise immunity.

Motorola scalable CAN (msCAN) — The Motorola scalable controller area network is a serial communications protocol that efficiently supports distributed real-time control with a very high level of data integrity.

msCAN — See "Motorola scalable CAN".

MSI — See "multiple serial interface".

multiple serial interface — A module consisting of multiple independent serial I/O sub-systems, e.g. two SCI and one SPI.

multiplexer — A device that can select one of a number of inputs and pass the logic level of that input on to the output.

nibble — A set of four bits (half of a byte).

object code — The output from an assembler or compiler that is itself executable machine code, or is suitable for processing to produce executable machine code.

opcode — A binary code that instructs the CPU to perform an operation.

open-drain — An output that has no pullup transistor. An external pullup device can be connected to the power supply to provide the logic 1 output voltage.

operand — Data on which an operation is performed. Usually a statement consists of an operator and an operand. For example, the operator may be an add instruction, and the operand may be the quantity to be added.

oscillator — A circuit that produces a constant frequency square wave that is used by the computer as a timing and sequencing reference.

OTPROM — One-time programmable read-only memory. A nonvolatile type of memory that cannot be reprogrammed.

overflow — A quantity that is too large to be contained in one byte or one word.

page zero — The first 256 bytes of memory (addresses $0000–$00FF).

parity — An error-checking scheme that counts the number of logic 1s in each byte transmitted. In a system that uses odd parity, every byte is expected to have an odd number of logic 1s. In an even parity system, every byte should have an even number of logic 1s. In the transmitter, a parity generator appends an extra bit to each byte to make the number of logic 1s odd for odd parity or even for even parity. A parity checker in the receiver counts the number of logic 1s in each byte. The parity checker generates an error signal if it finds a byte with an incorrect number of logic 1s.

PC — See “program counter (PC).”

peripheral — A circuit not under direct CPU control.

phase-locked loop (PLL) — A clock generator circuit in which a voltage controlled oscillator produces an oscillation which is synchronized to a reference signal.

PLL — See "phase-locked loop (PLL)."

pointer — Pointer register. An index register is sometimes called a pointer register because its contents are used in the calculation of the address of an operand, and therefore points to the operand.

polarity — The two opposite logic levels, logic 1 and logic 0, which correspond to two different voltage levels, V_{DD} and V_{SS}.

polling — Periodically reading a status bit to monitor the condition of a peripheral device.

port — A set of wires for communicating with off-chip devices.

prescaler — A circuit that generates an output signal related to the input signal by a fractional scale factor such as 1/2, 1/8, 1/10 etc.

program — A set of computer instructions that cause a computer to perform a desired operation or operations.

program counter (PC) — A 16-bit register in the CPU. The PC register holds the address of the next instruction or operand that the CPU will use.

pull — An instruction that copies into the accumulator the contents of a stack RAM location. The stack RAM address is in the stack pointer.

pullup — A transistor in the output of a logic gate that connects the output to the logic 1 voltage of the power supply.

pulse-width — The amount of time a signal is on as opposed to being in its off state.

pulse-width modulation (PWM) — Controlled variation (modulation) of the pulse width of a signal with a constant frequency.

push — An instruction that copies the contents of the accumulator to the stack RAM. The stack RAM address is in the stack pointer.

PWM period — The time required for one complete cycle of a PWM waveform.

RAM — Random access memory. All RAM locations can be read or written by the CPU. The contents of a RAM memory location remain valid until the CPU writes a different value or until power is turned off.

RC circuit — A circuit consisting of capacitors and resistors having a defined time constant.

read — To copy the contents of a memory location to the accumulator.

register — A circuit that stores a group of bits.

reserved memory location — A memory location that is used only in special factory test modes. Writing to a reserved location has no effect. Reading a reserved location returns an unpredictable value.

reset — To force a device to a known condition.

SCI — See "serial communication interface module (SCI)."

serial — Pertaining to sequential transmission over a single line.

serial communications interface module (SCI) — A module that supports asynchronous communication.

serial peripheral interface module (SPI) — A module that supports synchronous communication.

set — To change a bit from logic 0 to logic 1; opposite of clear.

shift register — A chain of circuits that can retain the logic levels (logic 1 or logic 0) written to them and that can shift the logic levels to the right or left through adjacent circuits in the chain.

signed — A binary number notation that accommodates both positive and negative numbers. The most significant bit is used to indicate whether the number is positive or negative, normally logic 0 for positive and logic 1 for negative. The other seven bits indicate the magnitude of the number.

software — Instructions and data that control the operation of a microcontroller.

software interrupt (SWI) — An instruction that causes an interrupt and its associated vector fetch.

SPI — See "serial peripheral interface module (SPI)."

stack — A portion of RAM reserved for storage of CPU register contents and subroutine return addresses.

stack pointer (SP) — A 16-bit register in the CPU containing the address of the next available storage location on the stack.

start bit — A bit that signals the beginning of an asynchronous serial transmission.

status bit — A register bit that indicates the condition of a device.

stop bit — A bit that signals the end of an asynchronous serial transmission.

subroutine — A sequence of instructions to be used more than once in the course of a program. The last instruction in a subroutine is a return from subroutine (RTS) instruction. At each place in the main program where the subroutine instructions are needed, a jump or branch to subroutine (JSR or BSR) instruction is used to call the subroutine. The CPU leaves the flow of the main program to execute the instructions in the subroutine. When the RTS instruction is executed, the CPU returns to the main program where it left off.

synchronous — Refers to logic circuits and operations that are synchronized by a common reference signal.

timer — A module used to relate events in a system to a point in time.

toggle — To change the state of an output from a logic 0 to a logic 1 or from a logic 1 to a logic 0.

tracking mode — A mode of PLL operation with narrow loop bandwidth. Also see ‘acquisition mode.’

two's complement — A means of performing binary subtraction using addition techniques. The most significant bit of a two's complement number indicates the sign of the number (1 indicates negative). The two's complement negative of a number is obtained by inverting each bit in the number and then adding 1 to the result.

unbuffered — Utilizes only one register for data; new data overwrites current data.

unimplemented memory location — A memory location that is not used. Writing to an unimplemented location has no effect. Reading an unimplemented location returns an unpredictable value.

variable — A value that changes during the course of program execution.

VCO — See "voltage-controlled oscillator."

vector — A memory location that contains the address of the beginning of a subroutine written to service an interrupt or reset.

voltage-controlled oscillator (VCO) — A circuit that produces an oscillating output signal of a frequency that is controlled by a dc voltage applied to a control input.

waveform — A graphical representation in which the amplitude of a wave is plotted against time.

wired-OR — Connection of circuit outputs so that if any output is high, the connection point is high.

word — A set of two bytes (16 bits).

write — The transfer of a byte of data from the CPU to a memory location.

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://motorola.com/semiconductors

Information in this document is provided solely to enable system and software implementers to use Motorola products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

MOTOROLA